General Linear Models (GLMS)

Marshall McQuillen

Contents

Linear Regression

Basic Form:

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j X_j$$

Each coefficient can be interpreted as the expected average change in the response Y for a one unit change in the predictor X_j, holding all other predictors $X_{j'}$ constant.

Assumptions

1. $f(x)$ **is linear in nature** - The true function $f(x)$ (or $\mathbf{E}(\mathbf{Y}|\mathbf{X})$, which can be read as, "The expected value of the response Y given the input array X.") is **linear in nature** (or that the linear assumption is a reasonable one). This means that, *regardless of the value of an attribute* X_j, a one unit increase in the value of X_j results in a β_j increase in Y (holding all other attributes constant).

   An example of this assumption being removed would be if there was a "leveling off" affect - where at a certain value of X_j, the expected absolute change in Y is *less* than what it was at a different level of X_j. Alternatively, there could be an "exponentiation" affect - where at a certain value of X_j, the expected absolute change in Y is *greater* than what it was at a different level of X_j.

   Either way, the the linear assumption states that β_j is the expected change in Y for **any value of** X_j.

2. $f(x)$ **is additive** - The additive assumption states that there aren't any interaction effects between the attributes. Another way of saying this is: "β_j, the expected change in Y for a one unit change in X_j, is independent of changes in any other attributes $X_{j'}$."

   Removing this assumption opens up the possibility for interaction terms, which is to say that changing the value of X_j could change $\beta_{j'}$, the effect of a different attribute $X_{j'}$ on the response Y.

   Adding an interaction term takes the form of:

$$
\begin{aligned}
y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 \\
&= \beta_0 + \beta_1 X_1 + \beta_3 X_1 X_2 + \beta_2 X_2 && \text{putting like terms near each other} \\
&= \beta_0 + X_1(\beta_1 + \beta_3 X_2) + \beta_2 X_2 && \text{combining like terms} \\
&= \beta_0 + \tilde{\beta}_1 X_1 + \beta_2 X_2 \quad \textit{where} \quad \tilde{\beta}_1 = \beta_1 + \beta_3 X_2 && \text{substitution}
\end{aligned}
$$

Thus, β_3 can be interpretted as the effect that a one unit change in X_2 has on the **effectiveness** of X_1 in changing Y.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import Pipeline

mod = LinearRegression()
mod.fit(x_train, y_train)
y_hat = mod.predict(x_test)

rmse = np.sqrt(mean_squared_error(y_test, y_hat))

lin_reg_pipe = Pipeline([
    ('column', ColumnSelector(name='column')),
    ('regression', LinearRegression())
])

y_hat = lin_reg_pipe.predict(x_test)
rmse = np.sqrt(mean_squared_error(y_test, y_hat))
```

Logistic Regression

Logistic regression is used in a classification setting, namely when the response is dichotomous (binary). When this is the case, one seeks to model the probability of an observation being a "success" (or "of interest", encoded as a 1) or a failure (encoded as a 0).

Logistic regression is similar to linear regression in that it still assumes the relationship between the attributes $\mathbf{X}$ and the response $\mathbf{y}$ is linear, however with one small caveat; logistic regression assumes that the log odds of a success are linear in $\mathbf{X}$.

Reasoning

Since the response in logistic regression is dichotomous, and one would like to model the probability of an observation being a success, a good place to start would be:

$$P(Y = 1|X = x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

$$P(Y = 1|X = x) = \beta_0 + \sum_{j=1}^{P} \beta_j x_j$$

However, this opens up the possibility of the output $P(Y = 1|X = x)$ taking on a value greater than 1 or less than 0, which violates probabilities. To remedy this, one makes use of the *logistic function* (hence the name),

which restricts all output to be within the bounds of 0 and 1, and therefore a valid probability (note that there are two ways to write the logistic function, but they are equivalent and will lead to the same result):

$$\frac{1}{1+\epsilon^{-x}} = f(x) = \frac{\epsilon^{x}}{\epsilon^{x}+1}$$

So, knowing that one would like the output to be a valid probability, we simply substitute $P(Y=1|X=x)$ into the above equation in place of $f(x)$, where x is equal to the linear model $\beta_0 + \sum_{j=1}^{P} \beta_j x_j$:

$$\frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}} = P(Y=1|X=x) = \frac{\epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j}}{\epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j}+1}$$

Quick Aside on Probability and Odds

In order to illustrate how everything comes together, it is important to note the relationship between the *probability* of an event occurring and the *odds* of an event occurring, given below:

$$Odds = \frac{P}{1-P} \quad and \quad P = \frac{Odds}{Odds+1}$$

Reasoning (continued)

Method 1

Now knowing the relationship between probabilities and odds, is clear how the right hand side of the two equations above are related, reproduced below:

$$\frac{Odds}{Odds+1} = P(Y=1|X=x) = \frac{\epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j}}{\epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j}+1}$$

Therefore, we can express the odds of a success as:

$$Odds = \epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j}$$

Using the natural logarithm on both sides of the equation, it is evident that **the log odds of a succes are linear in the attributes X**.

$$log(Odds) = log(\epsilon^{\beta_0+\sum_{j=1}^{P}\beta_j x_j})$$

$$log(Odds) = \beta_0 + \sum_{j=1}^{P} \beta_j x_j$$

Method 2

The other way to reconcile logistic regression is to look at the other expression of logistic regression, reproduced below:

$$P(Y=1|X=x) = \frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}}$$

Knowing the above, one can substitute the right side of the equation into the odds ratio and solve for $\beta_0 + \sum_{j=1}^{P}\beta_j x_j$:

$$Given \quad Odds = \frac{P}{1-P} \quad and \quad P(Y=1|X=x) = \frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}}$$

Then:

$$Odds = \frac{\left(\frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}}\right)}{1-\left(\frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}}\right)}$$

$$Odds = \frac{\frac{1}{a}}{1-\frac{1}{a}} \qquad \text{set } a = 1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)} \text{ to clean up this proof}$$

$$= \frac{\frac{1}{a}}{\frac{a}{a}-\frac{1}{a}} \qquad \text{change the 1 to } \frac{a}{a} \text{ to create like terms}$$

$$= \frac{\frac{a}{a}}{\frac{a^2}{a}-\frac{a}{a}} \qquad \text{multiply each term by } a$$

$$= \frac{1}{a-1} \qquad \text{simplify}$$

$$= \frac{1}{1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}-1} \qquad \text{substitute } 1+\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)} \text{ in for } a$$

$$= \frac{1}{\epsilon^{-(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}} \qquad \text{1's in denominator cancel}$$

$$= \epsilon^{(\beta_0+\sum_{j=1}^{P}\beta_j x_j)} \qquad \text{bring the negative exponent up}$$

$$Log(Odds) = Log(\epsilon^{(\beta_0+\sum_{j=1}^{P}\beta_j x_j)}) \qquad \text{take the log of both sides}$$

$$Log(Odds) = \beta_0 + \sum_{j=1}^{P}\beta_j x_j \qquad \text{the log odds are linear in } \mathbf{X}$$

And, once again, the conclusion is made that the log odds of a succes are linear in the attributes X.

$$Log(Odds) = \beta_0 + \sum_{j=1}^{P} \beta_j x_j$$

Loss Function

The loss function (or cost function) for logistic regression, called the log loss, is given below:

$$Log\ Loss = -\left(\frac{\sum_{i=1}^{N} (y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))}{N} \right)$$

Walkthrough

The key to understanding the loss function is to look at each term in the numerator, reproduced below:

$$Log\ loss = y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)$$

Since logistic regression is used in a *binary* setting, the response **y** will be a vector of 1's and 0's. Therefore, y_i will be either a 1 or a 0. With that in mind, it is clear that the first part of both terms in the above equation, y_i and $(1 - y_i)$, serve the same functionality that an indicator function serves; if y_i is a 1, the second term in the equation will zero out, and conversely, if y_i is a 0, the first term in the equation will zero out:

$$If \quad y_i = 1, \quad Log\ loss \begin{cases} = (1) log(\hat{y}_i) + (1 - 1) log(1 - \hat{y}_i) \\ = (1) log(\hat{y}_i) \\ = log(\hat{y}_i) \end{cases}$$

$$If \quad y_i = 0, \quad Log\ loss \begin{cases} = (0) log(\hat{y}_i) + (1 - 0) log(1 - \hat{y}_i) \\ = (1) log(1 - \hat{y}_i) \\ = log(1 - \hat{y}_i) \end{cases}$$

Now, the output of logistic regression is a **probability**, which is to say $\hat{y}_i$ will be the probability that observation i is a 1. Therefore, the loss function will always be taking the log of a probability. Referencing the formulas above, if $y_i = 1$, $Log\ loss = log(\hat{y}_i)$ and if $y_i = 0$, $Log\ loss = log(1 - \hat{y}_i)$.

Since all models seek to **minimize the loss**, ideally the log loss is as close to 0 as possible. As it turns out, $log(1) = 0$. So, when $y_i = 1$, $Log\ loss = log(1)$ will minimize the loss, and if $y_i = 0$, $Log\ loss = log(1 - 0)$ will minimize the loss. **It is clear that this setup minimizes the error when $\hat{y}_i$ is as close to y_i as possible, averaged over all observations**.

Interpretation

- Each coefficient of logistic regression is interpreted as the change in the **log of the odds** of a success for a one unit change in the predictor, holding all else constant. To translate this to a probability of a success:

1. Plug each predictor value of a test observation into the linear equation, using the respective coefficients and the intercept; this returns the **log odds** of a success for that test observation.

$$log(Odds) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 ...$$

2. Exponentiate this value; this removes the log and **returns the odds of a success** for the test observation.

$$Odds = \epsilon^{log(Odds)}$$

3. Divide the odds by one plus the odds (odds / (1 + odds)) and you have the **probability that the test observation is a success**.

$$Prob(y_i = 1) = \frac{Odds}{1 + Odds}$$

- *Note that your X matrix (NOT your y vector) must be scaled before being used by sklearn's LogisticRegression() class (it utilizes Lasso, which must have scaled data)*

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metric import accuracy_score, precision_score, recall_score

mod = LogisticRegression
mod.fit(x_train, y_train)
# return predicted probability of being a success
y_hat_probs = mod.predict_proba(x_test)[:, 1]
# returns the predicted class (success or failure)
y_hat = mod.predict(x_test)

acc = accuracy_score(y_test, y_hat)
prec = precision_score(y_test, y_hat)
recall = recall_score(y_test, y_hat)

def log_odds_to_prob(log_odds):
    """
    Converts the log odds of a success into the probability of a success.

    Input:
        log_odds: (int) The log odds of a success / the output of a logistic regression model
    Output:
        Probability: (float) The probability of a success
    """
    odds = np.exp(log_odds)
    prob = odds / (1 + odds)
    return prob
```

Regularized Regression

- Both Ridge Regression and Lasso Regression both model the relationship between a set **X** of p predictors and a quantitative response **y** as a linear model (same as linear regression).
- However, both Ridge and Lasso have an additional term appended onto the loss function of a linear model, the RSS:
  - Ridge: λ multiplied by the summation of the *squares* of all coefficients. ($l2$ penalty)

    * So, the loss function becomes

$$RSS + \lambda \sum_{j=1}^{p} \beta_i^2$$

- Lasso: λ multiplied by the summation of the *absolute values* of all the coefficients. ($l1$ penalty)
    * So, the loss function becomes

$$RSS + \lambda \sum_{j=1}^{p} |\beta_i|$$

The tuning parameter λ is best selected using cross validation. The practical difference between Lasso and Ridge regression is that Lasso will set some of the coefficients equal to *exactly zero* while ridge regression shrinks the coefficients *towards zero*.

Lasso Regression

```
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

lassie = Lasso()
lassie.fit(x_train, y_train)
y_hat = lassie.predict(x_test)
print(np.sqrt(mean_squared_error(y_test, y_hat)))
```

Ridge Regression

```
import numpy as np
from sklearn.linear_model import Ridge()
from sklearn.metrics import accuracy_score

ridge = Ridge()
ridge.fit(x_train, y_train)
y_hat = ridge.predict(x_test)
print(np.sqrt(accuracy_score(y_test, y_hat)))
```

Multicollinearity

Variance Inflation Factor (VIF)

Penn State VIF

The VIF is a statistic that can help one determine whether multicollinearity exists in multivariate regression. By regressing each of the k predictors on the remaining predictors, one can obtain an estimate of *how well the k_{th} predictor can be estimated with the other predictors*:

$$\textit{Instead of ...} \ \mathbf{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_p x_p$$
$$\textit{Build a model ...} \ \mathbf{x_j} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_{p-1} x_{p-1}$$

In the same manner that the R^2 value for a "normal" regression ($y = \beta_0 + \sum_{j=1}^{P} \beta_j x_j$) illustrates the percentage of the total variance in the response explained by the model, regressing the k_{th} variable on the other predictors can determine if the k_{th} predictor could be (somewhat accurately) predicted by the other predictors.

The VIF for the k_{th} predictor, VIF_k, is a measure of how much the variance of the coefficient for the k_{th} predictor is inflated due to the existence of multicollinearity. Looking at the equation below, it is clear that if $R_k^2 = 0$, which is to say there is zero multicollinearity between the k_{th} predictor and the other predictors, the VIF_k would be equal to 1. Alternatively, $VIF's$ exceeding 4 - 5 warrant further investigation, since that indcates there is some multicollinearity.

$$VIF_k = \frac{1}{1 - R_k^2}$$