Ridge Regression

J.R. Powers-Luhn

***Abstract*—A regularization method for linear regression is explored. A term is added to the cost function proportional to the magnitude of the coefficients. A scaling term, α, adjusts the tradeoffs between model bias and variance. Two methods for optimizing the scaling term are examined. This method is used to generate models for predicting body fat percentage. The selected model is accurate to a root mean squared error of 4.34.**

I. Introduction

Least squares regression makes the assumption that its predictors are linearly independent of each other. If the variables are not linearly independent, then the matrix of model inputs is singular and cannot be inverted. In the real world, the presence of random noise precludes any two inputs being perfectly correlated, but this also means that small changes in the input measurements (i.e. a slightly different noise value from the same distribution) can have large, unpredictable effects on the coefficients from the regression model. Since many coefficients fit the data to within the noise present in the system, it is difficult to choose the correct coefficients.

One method to address this is to apply a regularization term to the cost function. By applying prior knowledge of the system (in this case, the fact that inputs and outputs should vary smoothly and continuously) it is possible to select from the "many coefficients" above.

The regularization term applied in this case is the ℓ_2 norm of the coefficients of the model. When this term is appropriately scaled it allows a tradeoff between the variance and bias of the model. Due to similarities between this method and quadratic response functions this method is known as ridge regression. [1].

II. Methodology

The naive solution to the linear regression problem

$$\mathbf{X}\vec{b} = \vec{y}$$

is to invert X and left-multiply:

$$\vec{b} = \mathbf{X}^{-1}\vec{y},$$

but this depends on how invertible X is. This is expressed in the condition number of X (equation 1, where the λ values are the largest and smallest eigenvalues of $\mathbf{X}$). Large condition numbers (in excess of 100) imply that the matrix is nearly non-invertible and the solution will vary significantly based on the small variations caused by noise.

$$Cond(\mathbf{X}) = \frac{\lambda_{max}}{\lambda_{min}} \tag{1}$$

By assuming that the solution to the linear regression problem should be insensitive to noise, it follows that the coefficients $\vec{b}$ should be small. The cost function is thus modified to that shown in equation 2.

$$||\vec{y} - \mathbf{X}\vec{b}||_2 + \alpha^2||\vec{b}||_2 \tag{2}$$

The derivative of this cost function is taken in order to find the value of $\vec{b}$ that minimizes it, leading to (after multiplying the original equation by the transpose of $\mathbf{X}$ to get a square matrix) equation 3.

$$\vec{b} = (\mathbf{X}^T\mathbf{X} + \alpha^2\mathbf{I})^{-1}\mathbf{X}^T\vec{y} \tag{3}$$

This creates an inherent tradeoff between variance and bias. If α is too small, the model will have a high variance and be subject to noise in the data. If α is too large, the model will have a high bias–it will be very stable, but will not fit the data well. Several methods exist for selecting α; two are described below.

In order to examine this method, a simulated dataset was generated consisting of one hundred measurements each of two perfectly correlated input "measurements", x_1 and x_2. x_1 and x_2 differed in scale, but were otherwise identical. Uniformly distributed noise was added to each measurement. Output values were generated from the equation $y = 0.1x_1 + 0.9x_2$ with additional uniform noise added.

The data were first split into training, test, and validation sets. Each variable (corresponding to a column in the input matrix) is scaled to be mean centered with unit variance. The output variables are also scaled in the same way. Since the regularization term penalizes large coefficients, scaling the variables removes the dependence on the units and puts all the coefficients on the same playing field.

A. Leave One Out Cross Validation

One method of determining the best value for α is to apply equation 3 to all samples in the training set except one, which is used to determine the error. This is repeated with a different sample used to determine the error and the first sample returned to the training set. This process is repeated until every sample has been used to determine error once and used to solve for $\vec{b}$ $n-1$ times, where n is the number of samples in the training set. The error values from each calculation are averaged to produce a single value. This is repeated for each candidate value for α. The value of α which results in the smallest average error is selected as the best.

Applying this method yielded an α value of 0.24, corresponding to a test RMSE of 0.30.

B. L-curve

An optimal value for α can also be determined graphically. Ridge regression is a tradeoff between a desire for small

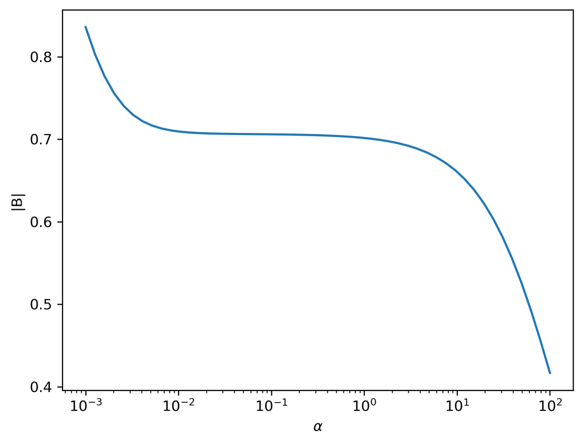


Fig. 1: Small values of $\alpha << \lambda_{min}$ allow large coefficients; α λ_{min} introduces a trade-off region with relatively stable models; values of $\alpha >> \lambda_{min}$ cause vanishingly small coefficients (and high bias).

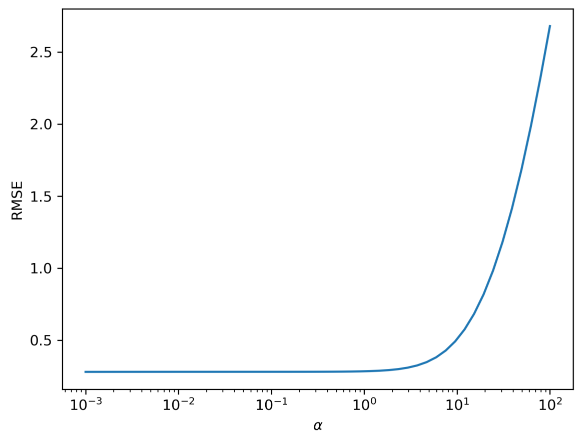


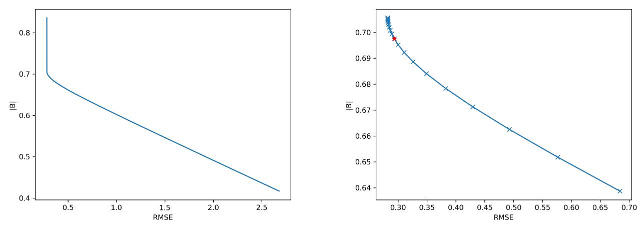
Fig. 2: α has little impact on RMSE until it nears the smallest eigenvalue of $\mathbf{X}^T\mathbf{X}$. When it exceeds this eigenvalue, the model becomes more biased and RMSE increases.

coefficients and small error. By plotting these terms against each other for a variety of α values it is possible to measure which α minimizes both of these.

To show this, ridge regression models were again generated for a variety of candidate α. The norm of the coefficients (figure 1) and the root mean squared error (2) for each model was calculated using the training set.

When the norm of the model coefficients and RMSE are plotted against each other (see figure 3) the point of closest approach to the origin can be measured. The α value corresponding to this point is selected for the best model.

Using this approach, an α value of 1.84 was selected. This model had a test error of 0.31.



(a) L_2 norm of coefficients vs RMSE (b) Close zoom on elbow region

Fig. 3: The point on the plot of $||\vec{b}||$ vs RMSE closest to the origin is the best compromise between model bias and variance.

C. Condition Number

Because these data were highly linearly dependent, the condition number was high, with a value of 2.1×10^6. Standardizing the data reduced this somewhat (to 2.6×10^5), but still high. By adding the α^2 term, the condition number becomes

$$Cond(\mathbf{X}) = \frac{\lambda_{max} + \alpha}{\lambda_{min} + \alpha}.$$

This disproportionately affects the denominator, pushing the condition number down. This also sets the scale of the α value–if $\alpha << \lambda_{min}$, ridge regression acts essentially like least squares regression. If $\alpha >> \lambda_{min}$, the model will have a high bias and will not fit the data well. α should be on a similar scale to the smallest eigenvalue of $\mathbf{X}^T\mathbf{X}$. With the α selected in section II-B, this results in a condition number of 79, below the rule-of-thumb value of 100.

D. Body Composition

A dataset of body measurements (body fat percentage, age, weight, height, adiposity index, and ten circumference measurements) was obtained [2]. Ridge regression was used to generate models to predict body fat percentage from the other predictor variables using the two methods described above to select the hyperparameter α.

III. Results

A. Leave One Out (LOO) Cross Validation

The value of α that minimized the mean RMSE from the cross-validated segments was determined to be 0.10. This corresponded to an RMSE of 3.83 on the test set.

B. L-curve

The value of α that minimized the ℓ_2 norm of $\vec{b}$ and RMSE was determined to be 2.81. This gave a test set error of 3.86.

C. Model Stability

Initially the matrix $\mathbf{X}^T\mathbf{X}$ had a condition number of 3.7×10^5. Standardizing the training data reduced this to 2056–a significant effect, likely because of the very different units and scales of the predictors. Applying the α value determined in section III-B reduced this an additional order of magnitude to 470, still not below the threshold value of 100.

D. Model Selection

The model that was selected was that produced by the L-curve method. Its performance was not significantly different from the model produced using the LOO method and the larger α value should result in greater stability. Both models significantly improved on the least squares model, which had an RMSE of 19.35.

The selected model was evaluated on the validation set data, producing an RMSE of 4.34.

IV. Conclusions

A model was generated to predict body fat percentage from body composition predictors. The error of this model was 4.34. This was comparable to the error produced by a partial least squares model (4.35) and showed improvement over principal component regression (4.72) and a best-guess linear regression (4.79). Further, the linear model required the calculation of inverse terms while the ridge regression was both straightforward to calculate (unlike PLS, no iterative algorithms were necessary) and did not require non-linear terms. Two methods of selecting the hyperparameter α were examined. Leave one out cross validation was easy to understand but computationally expensive and might be prohibitive for large datasets. Variations on this method (e.g. "Leave K out") are worth exploring if they lead to stable results. The L-curve method of determining α clearly demonstrates the bias/variance trade, making it attractive if potentially subjective.

References

[1] Arthur E Hoerl and Robert W Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[2] K W Penrose, A G Nelson, and A G Fisher. "Generalized Body Composition Prediction Equation For Men Using Simple Measurement Techniques". In: *Medicine & Science in Sports & Exercise* 17.2 (1985). ISSN: 0195-9131.

V. Appendix

Python code used to perform calculations and generate graphics.

```
import matplotlib.pyplot as plt
import numpy as np
import numpy.linalg as la
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, LeaveOneOut
from utilities import train_test_val_split, load_matlab_data, rmse
from sklearn.metrics import make_scorer

np.random.seed(42)

t = np.arange(1, 101)
x1 = np.sin(t) + 0.01 * np.random.rand(t.shape[0])
x2 = 10 * np.sin(t) + 0.01 * np.random.rand(t.shape[0])
x = np.hstack([x1.reshape((-1, 1)), x2.reshape((-1, 1))])
y = 0.1 * x[:,0] + 0.9 * x[:,1] + np.random.rand(t.shape[0])

xtr, ytr, xts, yts, xv, yv = train_test_val_split(x, y, seed=42)
ytr = ytr.reshape((-1, 1))
yts = yts.reshape((-1, 1))
yv = yv.reshape((-1, 1))

xscaler = StandardScaler().fit(xtr)
yscaler = StandardScaler().fit(ytr)

xs = xscaler.transform(xtr)
ys = yscaler.transform(ytr)

lin = LinearRegression().fit(xs, ys)

# Find best alpha value with Leave One Out Cross validation
alpha_params = np.logspace(-2, 2)
params = {'alpha': alpha_params}
ridge_loo = GridSearchCV(Ridge(fit_intercept=False), param_grid=params,
                         scoring=make_scorer(rmse, greater_is_better=False),
                         cv=LeaveOneOut(), iid=True)
ridge_loo.fit(xs, ys)
a_loo = ridge_loo.best_estimator_.alpha
print(f"Best alpha for leave one out was {a_loo}")

# Find best alpha value with 10-fold cross validation
alpha_params = np.logspace(-2, 2)
params = {'alpha': alpha_params}
ridge_cv10 = GridSearchCV(Ridge(fit_intercept=False), param_grid=params,
                         scoring=make_scorer(rmse, greater_is_better=False),
                         cv=10, iid=True)
ridge_cv10.fit(xs, ys)
a_cv10 = ridge_cv10.best_estimator_.alpha
print(f"Best alpha for 10-fold was {a_cv10}")

# Find best alpha value with 3-fold cross validation
alpha_params = np.logspace(-2, 2)
params = {'alpha': alpha_params}
ridge_cv3 = GridSearchCV(Ridge(fit_intercept=False), param_grid=params,
                         scoring=make_scorer(rmse, greater_is_better=False),
                          cv=3, iid=True)
ridge_cv3.fit(xs, ys)
a_cv3 = ridge_cv3.best_estimator_.alpha
print(f"Best alpha for 3-fold was {a_cv3}")

yts_pred_s = ridge_loo.best_estimator_.predict(xscaler.transform(xts))
sim_error_loo = rmse(yts, yscaler.inverse_transform(yts_pred_s))
print(f"Test error for LOO alpha: {sim_error_loo}")

def best_alpha(folds, x, y):
    alpha_params = np.logspace(-2, 2)
    params = {'alpha': alpha_params}
    r = GridSearchCV(Ridge(fit_intercept=False),
                     param_grid=params,
                     scoring=make_scorer(rmse, greater_is_better=False),
                     cv=folds, iid=True)
    r.fit(x, y)
    a = r.best_estimator_.alpha
    return a

folds = np.arange(2, xs.shape[0])
alphas = [best_alpha(f, xs, ys) for f in folds]
f = plt.figure()
plt.plot(folds, alphas)
plt.xlabel("Number of folds")
plt.ylabel(r"Optimal $\alpha$")
plt.savefig("images/alpha_vs_folds.png", dpi=300)
plt.clf()

# Now try L-curve
alpha_params = np.logspace(-3, 2)
# Generate a model for each alpha value
models = [Ridge(alpha=a, fit_intercept=False).fit(xs, ys) for a in alpha_params]
norm_b = [la.norm(m.coef_) for m in models]
ytr_scaled_pred = [m.predict(xscaler.transform(xtr)) for m in models]
rmse_vals = [rmse(ytr, yscaler.inverse_transform(y)) for y in ytr_scaled_pred]

# Examine norm of coefficients vs RMSE
plt.plot(rmse_vals, norm_b)
plt.ylabel("|B|")
plt.xlabel("RMSE")
plt.savefig("images/norm_b_vs_rmse.png", dpi=300)
plt.clf()

# Figure out where to zoom in
opt_b = np.array(rmse_vals) ** 2 + np.array(norm_b) ** 2
opt_b_loc = np.where(opt_b == np.min(opt_b))[0][0]
start_ind = max([0, opt_b_loc -10])
end_ind = min([opt_b.shape[0], opt_b_loc+10])
plt.plot(rmse_vals[start_ind:end_ind], norm_b[start_ind:end_ind], 'x-')
plt.plot(rmse_vals[opt_b_loc], norm_b[opt_b_loc], 'r*')
plt.ylabel("|B|")
plt.xlabel("RMSE")
plt.savefig("images/norm_b_vs_rmse_zoomed.png", dpi=300)
plt.clf()

# Look at effect of increasing alpha on RMSE
# Pretty evident that bias increases as alpha goes up
```

```
plt.plot(alpha_params, rmse_vals)
plt.xscale('log')
plt.xlabel(r"$\alpha$")
plt.ylabel("RMSE")
plt.savefig("images/rmse_vs_alpha.png", dpi=300)
plt.clf()

# Look at effect of increasing alpha on coefficient norm
plt.plot(alpha_params, norm_b)
plt.xscale('log')
plt.xlabel(r"$\alpha$")
plt.ylabel("|B|")
plt.savefig("images/norm_b_vs_alpha.png", dpi=300)
plt.clf()

# Examine the stability of the result
def cond_ridge(x, alpha):
    e = np.eye(x.shape[1])
    e = e * alpha
    c = la.cond(x.T @ x + e)
    return c

# Effect of alpha on condition number
plt.plot(alpha_params, [cond_ridge(xs, a) for a in alpha_params])
plt.xscale('log')
plt.xlabel(r"$\alpha$")
plt.ylabel("Condition number")
plt.savefig("images/condition_number_vs_alpha.png", dpi=300)
plt.clf()

print(f"Best alpha from L-curve method: {alpha_params[opt_b_loc]}")

# Original training matrix
c_unscaled = cond_ridge(xtr, 0)
print(f"Condition number of unscaled matrix: {c_unscaled}")

# Scaled training matrix
c_scaled = cond_ridge(xs, 0)
print(f"Condition number of scaled matrix: {c_scaled}")

# Ridge regression condition number
c_ridge = cond_ridge(xs, alpha_params[opt_b_loc])
print(f"Condition number for ridge regression: {c_ridge}")

# Compare model performance
lin_predictions = yscaler.inverse_transform(
    lin.predict(xscaler.transform(xts))
)
linear_performance = rmse(yts, lin_predictions)
print(f"Linear regression RMSE: {linear_performance}")

ridge_lc = Ridge(alpha=alpha_params[opt_b_loc],
                 fit_intercept=False).fit(xs, ys)
ridge_lc_predictions = yscaler.inverse_transform(
    ridge_lc.predict(xscaler.transform(xts))
)
ridge_performance = rmse(yts, ridge_lc_predictions)
print(f"Ridge regression RMSE: {ridge_performance}")
print("—————————————————————————————")
print("Now changing from simulated to real data")
print("—————————————————————————————")

# Load body weight data
x, y = load_matlab_data("data/hwkdataNEW.mat")
xtr, ytr, xts, yts, xv, yv = train_test_val_split(x, y, seed=3)
ytr = ytr.reshape((-1, 1))
yts = yts.reshape((-1, 1))
yv = yv.reshape((-1, 1))

# Fit new scalers to that data
xscaler = StandardScaler().fit(xtr)
yscaler = StandardScaler().fit(ytr)
xs = xscaler.transform(xtr)
ys = yscaler.transform(ytr)

# Create and fit a linear regression model for comparison
lin = LinearRegression(fit_intercept=False).fit(xs, ys)
x_test_scaled = xscaler.transform(xts)
y_test_scaled = yscaler.transform(yts)
lin_error = rmse(yts, lin.predict(x_test_scaled))
print(f"Error for linear regression: {lin_error}")

# Use two methods to find the best value for alpha

# First, Leave One Out Cross Validation
alpha_parameters = np.logspace(-2, 3)
params = {'alpha': alpha_parameters}
ridge_loo = GridSearchCV(Ridge(fit_intercept=False),
                         param_grid=params, cv=LeaveOneOut(),
                         scoring=make_scorer(rmse, greater_is_better=False),
                         iid=True)
ridge_loo.fit(xs, ys)
print(f"Best alpha from Leave one out: {ridge_loo.best_params_}")

plt.plot(ridge_loo.cv_results_['param_alpha'],
         -ridge_loo.cv_results_['mean_test_score'])
plt.plot(ridge_loo.best_params_['alpha'],
         -ridge_loo.best_score_, 'ro')
plt.xlabel(r"$\alpha$")
plt.ylabel("RMSE")
plt.xscale("log")
plt.savefig("images/bw_rmse_vs_alpha.png", dpi=300)
plt.clf()

# L-curve method

# Fit models for every alpha value
models = [Ridge(alpha=a, fit_intercept=False).fit(xs, ys) for a in alpha_parameters]

# Calculate parameters for those models
norm_b = [la.norm(m.coef_) for m in models]
rmse_vs_alpha = [rmse(yts, yscaler.inverse_transform(m.predict(x_test_scaled))) \
                 for m in models]

rmse_vs_alpha = [rmse(ytr, yscaler.inverse_transform(m.predict(xs))) \
                     for m in models]

plt.plot(rmse_vs_alpha, norm_b)
plt.xlabel("RMSE")
plt.ylabel("|B|")
plt.savefig("images/bw_norm_b_vs_rmse.png", dpi=300)
plt.clf()

# Find the point closest to the origin
paired_points = np.array([_ for _ in zip(rmse_vs_alpha, norm_b)])

min_norm_ind = np.where(
    la.norm(paired_points, axis=1) == np.min(la.norm(paired_points, axis=1))
)[0][0]
print(f"Minimum distance from origin is at index {min_norm_ind}")
a = alpha_parameters[min_norm_ind]
print(f"The best alpha (L-curve) is {a}")

plt.plot(rmse_vs_alpha, norm_b, "x-")
plt.plot(rmse_vs_alpha[min_norm_ind], norm_b[min_norm_ind], 'ro')
plt.xlabel("RMSE")
plt.ylabel("|B|")
plt.savefig("images/bw_norm_b_vs_rmse_with_best_alpha.png", dpi=300)

ridge_lc = Ridge(alpha=a, fit_intercept=False).fit(xs, ys)

# Compare the two with linear
error_loo = rmse(yts, yscaler.inverse_transform(ridge_loo.predict(x_test_scaled)))
error_lc = rmse(yts, yscaler.inverse_transform(ridge_lc.predict(x_test_scaled)))
print(f"Leave one out CV\talpha: {ridge_loo.best_params_['alpha']:0.2f}"
      f"\t\tRMSE: {error_loo:0.2f}")
print(f"L-curve method\t\talpha: {ridge_lc.alpha:0.2f}\t\tRMSE: {error_lc:0.2f}")
print(f"Linear regression\t\t\t\tRMSE: {lin_error:0.2f}")

# Examine condition number

# Training matrix condition number
print(f"Condition no. of training matrix: {cond_ridge(xtr, 0):0.2f}")

# Scaled training matrix condition number
print(f"Condition no. of scaled training matrix: {cond_ridge(xs, 0):0.2f}")

print(f"Condition no. for ridge regression: {cond_ridge(xs, ridge_lc.alpha):0.2f}")

print("--------------------------------------------------")
print("Final model selection: L-curve")
print("--------------------------------------------------")
xvs = xscaler.transform(xv)
yvs_pred = ridge_lc.predict(xvs)
ridge_lc_val_error = rmse(yv, yscaler.inverse_transform(yvs_pred))
print(f"Final ridge model validation error: {ridge_lc_val_error:0.2f}")
```