Statistical Learning
Professor Keene

Mini-Project #2: Linear Regression

By Miraj Patel, Max Howald, and Frank Longueira

The dataset used in this project was found in the UCI Machine Learning Repository. It can be found at the following link: https://archive.ics.uci.edu/ml/datasets/Auto+MPG

The dataset contained 398 samples. Each sample was a record of a unique car with 7 features (discrete/continuous) & 1 continuous label.

Features:

1) 'cylinders' - number of cylinders inside engine (discrete)
2) 'displacement' – volume of engine's cylinders (continuous)
3) 'horsepower' – rate of energy consumption (continuous)
4) 'weight' – the weight of the car (continuous)
5) 'acceleration' – maximum acceleration of car (continuous)
6) 'model year' – year it was made (discrete)
7) 'origin' – country it was built in (discrete)

Label:

1) 'mpg' – miles per gallon rating of the car

Using these 7 features, our goal was to use linear regression, best k-subset, & shrinkage methods to develop a good model for predicting the continuous label 'mpg' (miles per gallon).

To start, we needed to clean up the data & get rid of 6 records that had missing fields This left us with 392 samples in the dataset. Next, since we are applying regression models, we normalized the dataset by scaling each feature/label column using its mean & standard deviation. This normalization put each column on the same magnitude & ensured no single feature would skew the model. After this, we split the dataset into a training set of 300 samples and a testing set of 92 samples. Using the training set, we computed the correlation matrix of the features and fit different models such as linear regression (OLS), best k-subset (for feature selection) then linear regression, ridge regression & lasso regression. All of this was done using Python. We coded up linear regression & finding zscores in order to practice coding up the algorithm, but we used the module "sklearn" for best-k subset, ridge regression, and lasso regression.

The correlation matrix provided some insight into which features were correlated with one another, such as weight & horsepower. Computation of z-scores allowed us to get even further insight into which features contributed most to the linear regression model. We noticed 'weight' and 'model year' had the highest absolute value of z-scores. This makes sense because how heavy the car is & when the car was made (advances in technology) should definitely play a large role into the 'mpg' of the car. The results of our linear regression model (using all the features) can be found in both Table 3.1 and Table 3.2 below, including our average mean squared error on the test set.

Following our textbook's prostate example, we then searched for the best subset of two features. We used a built-in Python function that scored features based on their F-values & we found that 'displacement' & 'weight' were chosen to form the best subset of two features. After this feature selection, we applied a linear regression model using only these two features. The results of our k-best linear regression model can be found in Table 3.3.

Finally, we applied ridge & lasso regression for a range of regularization parameter values. These shrinkage methods allow for a "shrinking" of coefficients by including a penalty term to the RSS in order to avoid overfitting to the training set. We have plots below for both ridge & lasso regression that show how the coefficients change with respect to the regularization parameter (lambda). In addition, we included an extra plot for each ridge & lasso regression which shows how the average mean squared error (test error) changes with respect to the regularization parameter (lambda). In Table 3.3, the reader can see which coefficients "shrunk" & which were left out when the models were trained on the training set. The results in this table look at a specific regularization value for each ridge & lasso regression. These values were estimated by looking at where the MSE (test error) began to spike for both ridge & lasso regression. We noticed the testing error went up with these two models & concluded that the full linear regression model (OLS) seemed to be the best one with respect to mean squared error.

TABLE 3.1. Correlations of predictors in the "auto-mpg" data

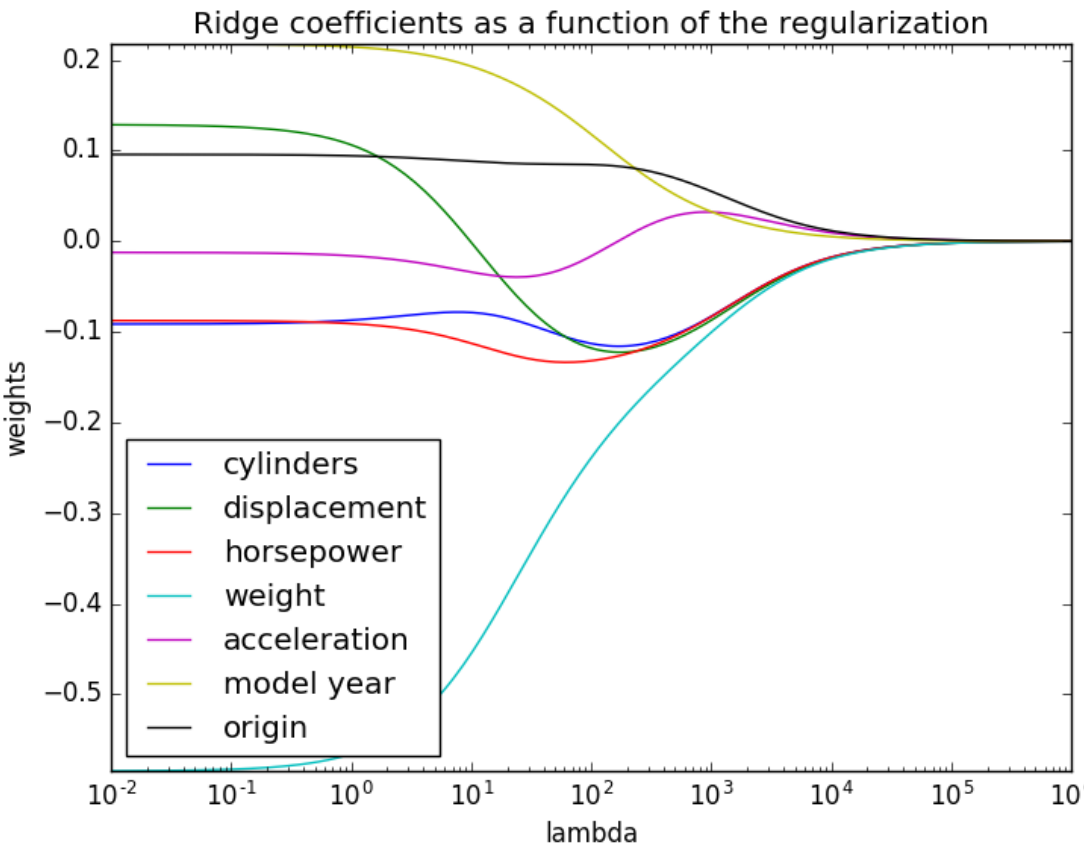
	cylinders	displacement	horsepower	weight	acceleration	model year
displacement	0.952					
horsepower	0.836	0.895				
weight	0.901	0.930	0.859			
acceleration	-0.539	-0.586	-0.719	-0.456		
model year	-0.126	-0.182	-0.269	-0.102	0.261	
origin	-0.629	-0.653	-0.465	-0.612	0.238	0.036

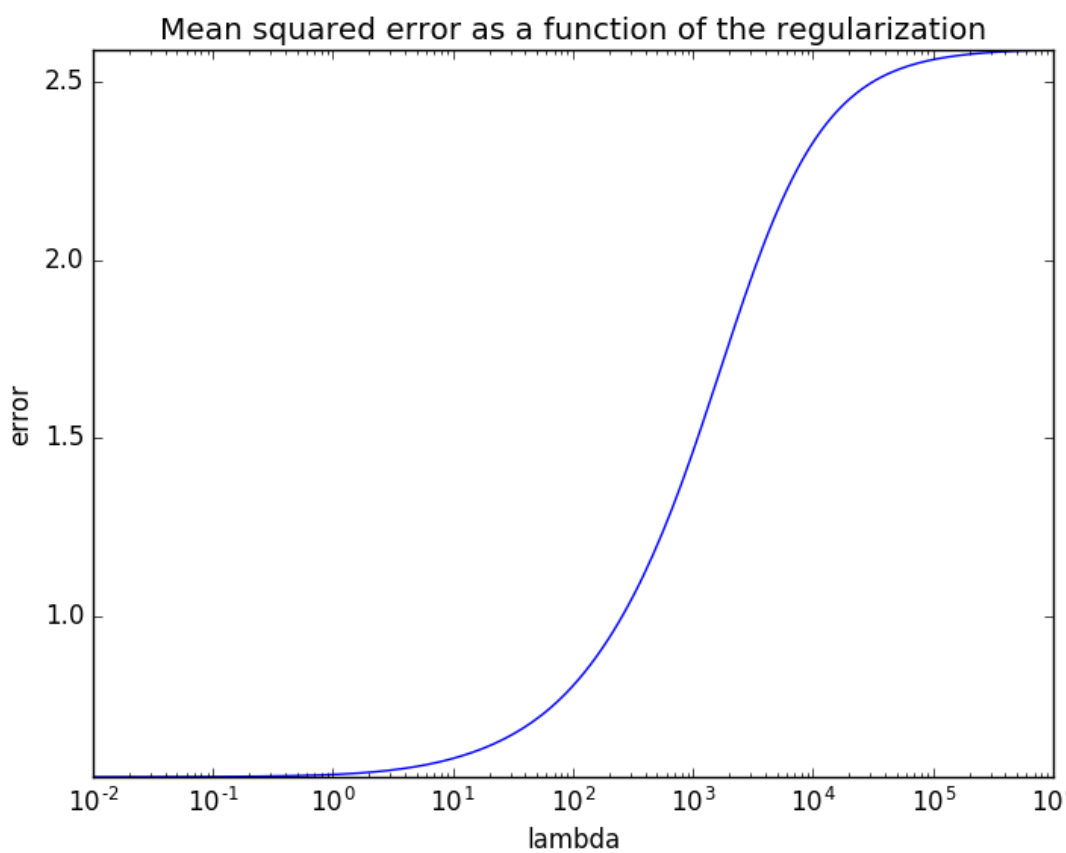
TABLE 3.2. Linear fit to the "auto-mpg" data

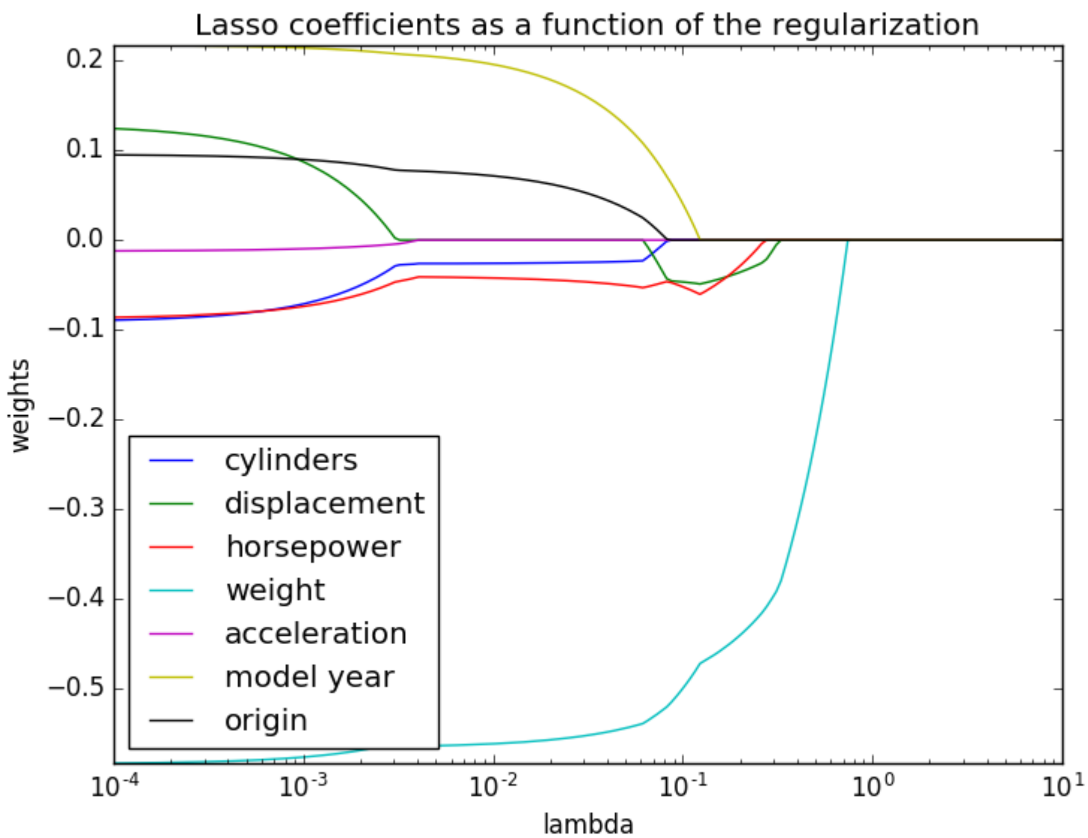
Term	Coefficient	Std. Error	Z Score
Intercept	-0.116	0.023	-4.990
cylinders	-0.092	0.065	-1.405
displacement	0.129	0.091	1.418
horsepower	-0.088	0.060	-1.465
weight	-0.585	0.062	-9.378
acceleration	-0.012	0.033	-0.374
model year	0.218	0.029	7.596
origin	0.096	0.031	3.087

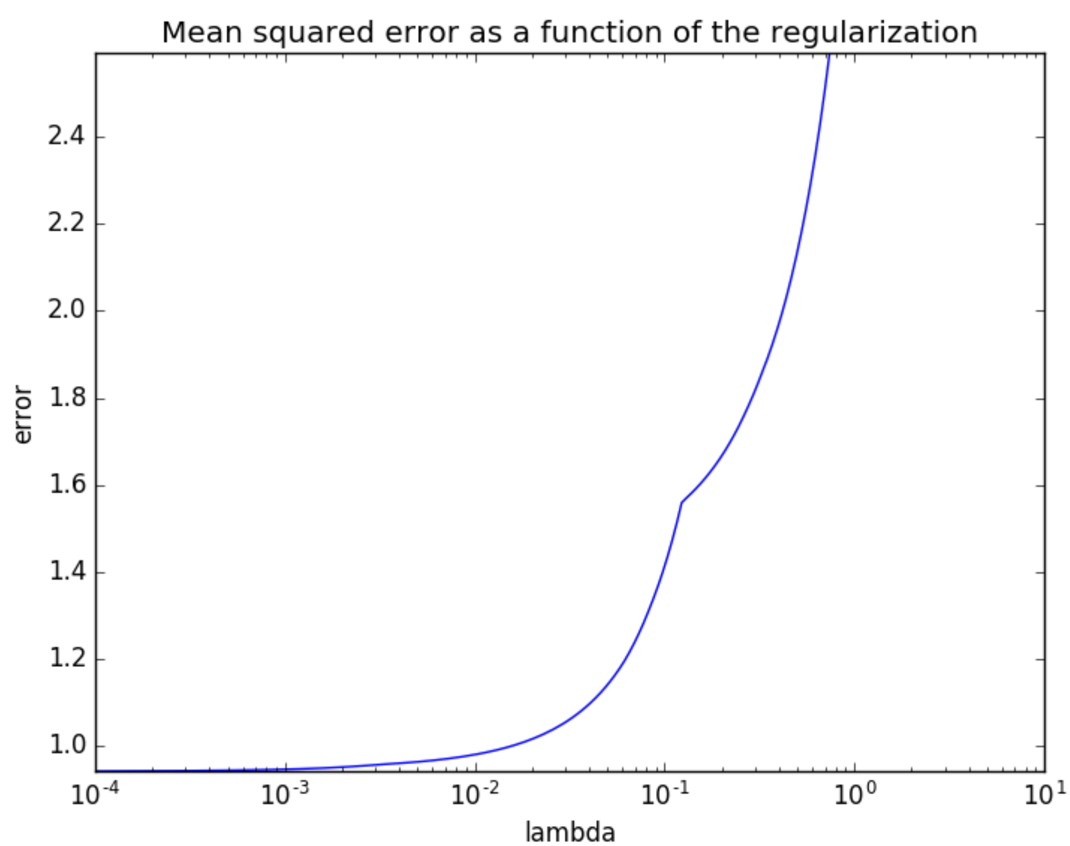
TABLE 3.3. Estimated coefficients & test error results, for different subset and shrinkage methods applied to the "auto-mpg" data. The blank entries correspond to variables omitted.

Term	LS	Best Subset ($k = 2$)	Ridge ($\lambda = 105$)	Lasso ($\lambda = 0.01$)
Intercept	-0.116	-0.204	0.163	-0.334
cylinders	-0.092		-0.113	-0.026
displacement	0.129	-0.150	-0.119	
horsepower	-0.088		-0.132	-0.042
weight	-0.585	-0.545	-0.236	-0.562
acceleration	-0.012		-0.015	
model year	0.218		0.116	0.196
origin	0.096		0.084	0.070
Test Error (MSE)	0.550	1.08	0.813	0.980


Ridge coefficients as a function of the regularization
weights
lambda
cylinders
displacement
horsepower
weight
acceleration
model year
origin
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6}


Mean squared error as a function of the regularization
error
lambda
2.5
2.0
1.5
1.0
10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6}


Lasso coefficients as a function of the regularization
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
weights
cylinders
displacement
horsepower
weight
acceleration
model year
origin
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
lambda


Mean squared error as a function of the regularization
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
error
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
lambda

Appendix (Code)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import Ridge, Lasso
from sklearn.preprocessing import scale
from operator import itemgetter

def linear_regression(X_train, Y_train) :
	# Compute number of observations & features from training data
	num_obs = np.shape(X_train)[0]
	num_features = np.shape(X_train)[1]

	# Create column of 1's equal in size to the number of observations
	ones_col = (np.ones(num_obs).T).reshape((-1,1))

	# Append column of 1's on the left of the features matrix
	X = np.concatenate((ones_col, X_train), axis = 1)

	X_T_X_inv = np.linalg.inv(np.dot((X.T), X))

	# Compute closed form solution for linear regression parameters
	B_hat = np.dot(np.dot((X_T_X_inv), X.T), Y_train)

	return B_hat

def linear_regression_predict(X_test, Y_test, B_hat):
	# Compute number of observations & features from training data
	num_obs = np.shape(X_test)[0]
	num_features = np.shape(X_test)[1]

	# Create column of 1's equal in size to the number of observations
	ones_col = (np.ones(num_obs).T).reshape((-1,1))
	X = np.concatenate((ones_col, X_test), axis = 1)

	Y_hat = np.dot(X, B_hat)
	avg_test_error = np.mean(np.square(Y_test-Y_hat))
	return avg_test_error


def find_zscores(X_train, Y_train, B_hat):

	# Compute number of observations & features from training data
	num_obs = np.shape(X_train)[0]
	num_features = np.shape(X_train)[1]

	# Create column of 1's equal in size to the number of observations
	ones_col = (np.ones(num_obs).T).reshape((-1,1))
```

```
	# Append column of 1's on the left of the features matrix
	X = np.concatenate((ones_col, X_train), axis = 1)

	# Compute fitted values at the training inputs
	Y_train_hat = np.dot(X, B_hat)

	# Estimate standard deviation of Y_train
	sigma_hat = np.sqrt((np.sum(np.square(Y_train-Y_train_hat)))/(num_obs -
num_features - 1))

	X_T_X_inv = np.linalg.inv(np.dot((X.T), X))

	# Extract diagonal entires
	v = np.diag(X_T_X_inv).reshape(-1,1)

	# Compute standard error for linear regression
	std_error = np.sqrt(v)*sigma_hat

	# Compute z_scores for linear regression
	z_scores = np.divide(B_hat, std_error)

	return std_error, z_scores


def correlationMatrix(X_train):
	corr_mat = np.corrcoef(X_train, rowvar = 0)
	return corr_mat



## Main script is below & calls the above functions

# Load in feature matrix & labels from dataset
num_features = 7
num_obs = 300
feature_names = ['cylinders', 'displacement', 'horsepower',	'weight',
'acceleration', 'model year',  'origin']

# Load in data and normalize it
mpg_data = scale(np.genfromtxt('auto-mpg.data.csv', delimiter=',', skip_header = 1,
dtype = 'float64', usecols = (0,1,2,3,4,5,6,7)))

# Separate training & testing data
X_train = mpg_data[0:num_obs, 0:num_features]
Y_train = mpg_data[0:num_obs, 7].reshape(-1, 1)

X_test = mpg_data[num_obs:, 0:num_features]
```

```
Y_test = mpg_data[num_obs:, 7].reshape(-1, 1)

# Compute correlation matrix of features from training data
corr_mat = correlationMatrix(X_train)

# Perform linear regression & obtain beta parameters for further predictions (using
all features)
B_hat = linear_regression(X_train, Y_train)

linear_regression_predict(X_test, Y_test, B_hat)

# Find zscores of the features
std_error, zscores = find_zscores(X_train, Y_train, B_hat)
table3_1 = np.concatenate((B_hat, std_error, zscores), axis = 1)

# Perform k-best subset feature selection using an F-value metric
kbest = 2
best_sub_model =  SelectKBest(f_regression, k = kbest)
X_train_kbest = best_sub_model.fit_transform(X_train, np.ravel(Y_train))


# Map features to their scores as determined by F-Values
kbest_scores = {}
for feature_name, score in zip(feature_names, best_sub_model.scores_):
	kbest_scores[feature_name] = score

kbest_Features = sorted(kbest_scores, key = kbest_scores.get, reverse=True)[:kbest]


# Perform linear regression & obtain beta parameters for further predictions (using
k-best features)
B_hat_kbest = linear_regression(X_train_kbest, Y_train)

linear_regression_predict(best_sub_model.transform(X_test), Y_test, B_hat_kbest)



# Train the ridge regression model with different regularisation strengths
lambdas = np.logspace(-2, 6, 200)
clf1 = Ridge()
coefs = []
intercepts = []
errors = []

for a in lambdas:
	clf1.set_params(alpha=a)
	clf1.fit(X_train, Y_train)
	coefs.append(clf1.coef_[0])
```

```
        intercepts.append(clf1.intercept_[0])
        Y_hat = clf1.predict(X_test)
        avg_test_error = np.mean(np.square(Y_test-Y_hat))
        errors.append(avg_test_error)

# Found by looking at plot
chosen_lambda_pos =  100
lambdas[chosen_lambda_pos]
intercepts[chosen_lambda_pos]
coefs[chosen_lambda_pos]
errors[chosen_lambda_pos]


# Display results
plt.figure(figsize=(20, 6))
plt.subplot(121)
ax = plt.gca()
ax.plot(lambdas, coefs)
ax.set_xscale('log')
ax.legend(feature_names, loc = 3)
plt.xlabel('lambda')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')

plt.subplot(122)
ax = plt.gca()
ax.plot(lambdas, errors)
ax.set_xscale('log')
plt.xlabel('lambda')
plt.ylabel('error')
plt.title('Mean squared error as a function of the regularization')
plt.axis('tight')
plt.show()


# Train the lasso regression model with different regularisation strengths
lambdas = np.logspace(-4, 1, 200)
clf2 = Lasso()
coefs = []
errors = []
intercepts = []

for a in lambdas:
        clf2.set_params(alpha=a)
        clf2.fit(X_train, Y_train)
        coefs.append(clf2.coef_)
        intercepts.append(clf1.intercept_[0])
```

```
        Y_hat = clf2.predict(X_test)
        avg_test_error = np.mean(np.square(Y_test-Y_hat))
        errors.append(avg_test_error)

# Found by looking at plot
chosen_lambda_pos =  80
lambdas[chosen_lambda_pos]
intercepts[chosen_lambda_pos]
coefs[chosen_lambda_pos]
errors[chosen_lambda_pos]

# Display results
plt.figure(figsize=(20, 6))

plt.subplot(121)
ax = plt.gca()
ax.plot(lambdas, coefs)
ax.set_xscale('log')
ax.legend(feature_names, loc = 3)
plt.xlabel('lambda')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization')
plt.axis('tight')

plt.subplot(122)
ax = plt.gca()
ax.plot(lambdas, errors)
ax.set_xscale('log')
plt.xlabel('lambda')
plt.ylabel('error')
plt.title('Mean squared error as a function of the regularization')
plt.axis('tight')
plt.show()
```