Regression regularization methods: Ridge, LASSO, and ElasticNet regression

Supervised parametric machine learning methods to predict a target variable.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score
```

Data simulation

Simulate some data distributed around the following linear relationship:

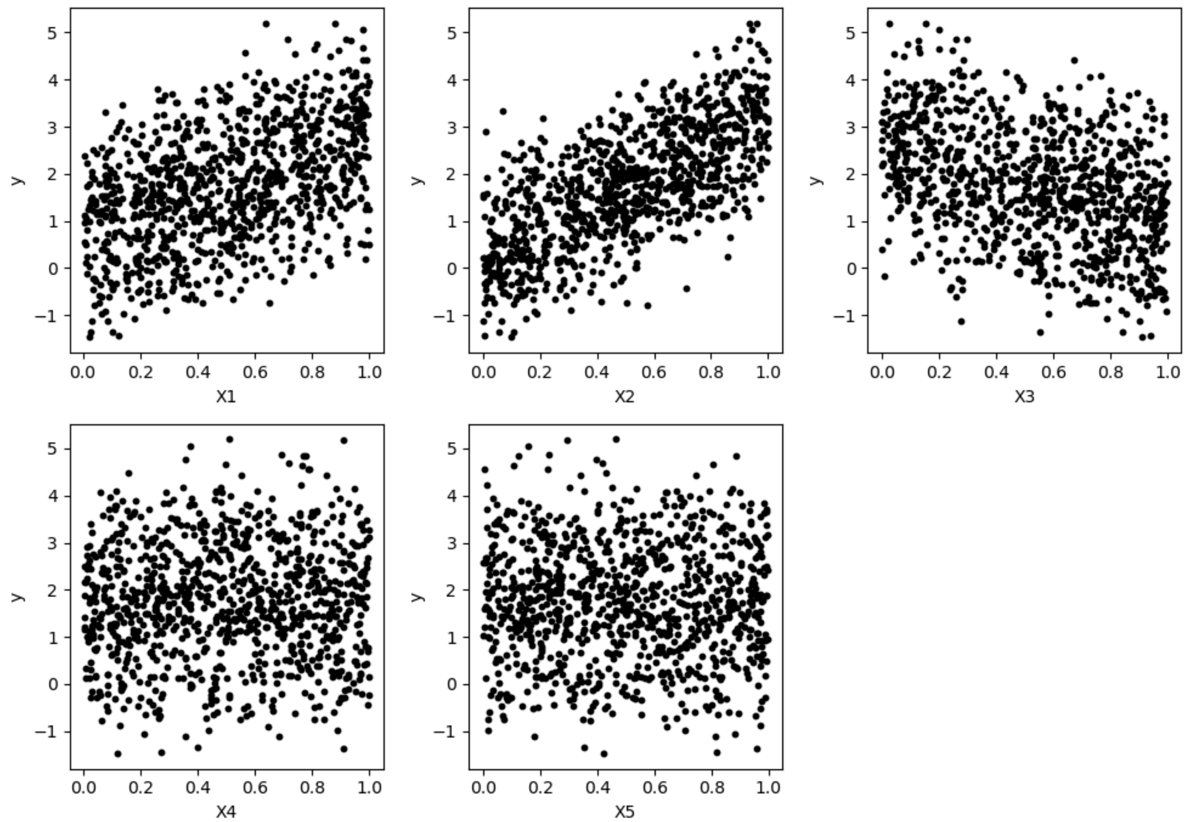
$$y = 2X_1 + 3X_2 - 1.5X_3 + 0X_4 + 0X_5 + \epsilon_i$$

In [2]:
```
#simulate data
n_samples = 1000
n_features = 5
X = np.random.rand(n_samples, n_features)
y = 2 * X[:, 0] + 3 * X[:, 1] - 1.5 * X[:, 2] + np.random.randn(n_samples) * 0.5
```

Visualize the data

Visualize the linear relationships in the simulated data.

In [3]:
```
#define figure space
fig = plt.figure(figsize = (10,7))
cols = 3
rows = 2

#create subplots
for i in range(1, n_features + 1):
    fig.add_subplot(rows, cols, i)
    plt.scatter(X[:, i - 1], y, s = 10, c = 'black')
    plt.xlabel(f"X{i}")
    plt.ylabel("y")

plt.tight_layout()
plt.show()
```



As expected, there exists clear linear relationships between the target variable (y) and feature variables (X1, X2, and X3), whereas X4 and X5 show a flat relationship.

Data proprocessing

Train and test split

Split the data into train and test datasets. This is done before to scaling to prevent data leakage.

In [4]:
```
#split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
```

Data scaling

There are two reasons for scaling prior to regularization:

1. normalization of variables to prevent unfair penaltizations across variables with different scales.
2. improved convergence to estimate parameters (train) more quickly and efficiently.

In [5]:
```
#scale the feature variables
scale = StandardScaler()
```

```
X_train_scaled = scale.fit_transform(X_train)
X_test_scaled = scale.fit_transform(X_test)
```

Model fitting

Fitting Ridge, LASSO, and ElasticNet regression models.

In [6]:
```
import random

#define function to split data into k-folds for cross-validation
def k_folds_split(df, k):
    #test set size
    test_size = len(df) // k

    #data frame indices
    df_idx = list(range(len(df)))
    unsampled_idx = df_idx

    #split train and test sets
    train_test_sets = []
    for _ in range(k):
        #train and test set indices
        test_idx = random.sample(unsampled_idx, test_size)
        train_idx = [x for x in df_idx if x not in test_idx]
        unsampled_idx = [x for x in unsampled_idx if x not in test_idx]

        #create train and test sets
        test_df = df.iloc[test_idx]
        train_df = df.iloc[train_idx]
        train_test_sets.append([test_df, train_df])

    return train_test_sets
```

In [7]:
```
#define the range of penalty terms to test
alphas = 10**np.linspace(-2, 10, 100)
```

Ridge regression

Sqaured shrinkage penalty:

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

Lambda penalty optimization

Optimize the penalty term using k-fold cross-validation.

In [8]:
```
#concatenate the training data target and feature variables
ridge_df = pd.DataFrame(np.hstack((y_train.reshape(-1,1), X_train_scaled)))
```

```
#split the training data into k-folds
cv = 5
ridge_cv = k_folds_split(ridge_df, cv)
len(ridge_cv)
```

Out[8]: 5

In [9]:
```
#define the ridge regression model
ridge = Ridge(fit_intercept = True)
ridge_results = []

for alpha in alphas:
    #set the penalty term
    ridge.set_params(alpha = alpha)

    #for each CV fold
    for i in range(cv):
        #define train and test sets
        X_train_cv = ridge_cv[i][1].drop(0, axis = 1)
        y_train_cv = ridge_cv[i][1][0]
        X_test_cv = ridge_cv[i][0].drop(0, axis = 1)
        y_test_cv = ridge_cv[i][0][0]

        #fit the ridge regression model
        ridge.fit(X_train_cv, y_train_cv)

        #calculate test scores (test MSE and R-squared)
        mse = mean_squared_error(y_test_cv, ridge.predict(X_test_cv))
        r2 = r2_score(y_test_cv, ridge.predict(X_test_cv))

        #store the results
        ridge_results.append([alpha, i + 1, mse, r2, ridge.coef_])

ridge_results = pd.DataFrame(ridge_results)
ridge_results.columns = ['alpha','fold','mse','r2','coefs']
ridge_results.head()
```

Out[9]:

	alpha	fold	mse	r2	coefs
0	0.01	1	0.247314	0.844683	[0.5919266840393018, 0.8490816818193622, -0.46...
1	0.01	2	0.225836	0.815703	[0.5662607150465646, 0.8580066198468511, -0.47...
2	0.01	3	0.244017	0.844206	[0.5787252044108461, 0.8362798854129531, -0.45...
3	0.01	4	0.264253	0.835607	[0.5803607941733478, 0.841400810807215, -0.469...
4	0.01	5	0.234303	0.850742	[0.5888676595857455, 0.8552120933182095, -0.45...

Determine the optimal penalty term for the Ridge regression model.

In [10]:
```
#average the test MSE across each CV fold
ridge_avg = pd.DataFrame(ridge_results.drop('coefs', axis = 1).groupby('alpha').mea
ridge_avg.reset_index(inplace = True)

#determine the penalty term with the lowest test MSE
```

```
min_idx = ridge_avg['mse'].idxmin()
ridge_alpha = ridge_avg.at[min_idx, 'alpha']
print(f"Ridge optimal penalty term: {ridge_alpha}")
```

```
Ridge optimal penalty term: 2.656087782946687
```

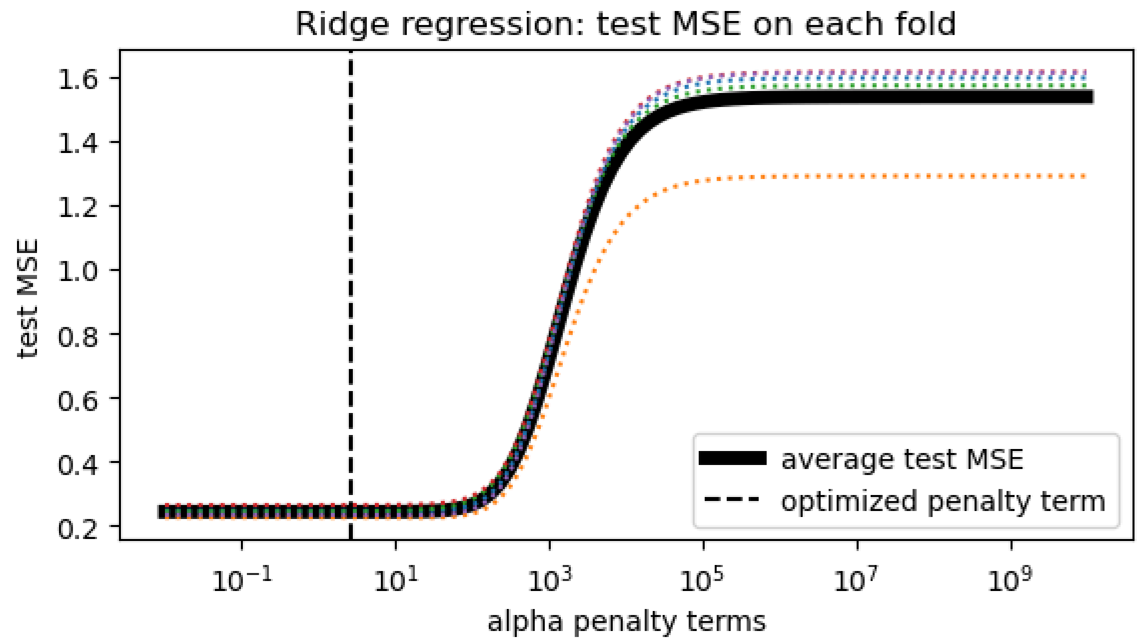
Visualize model performance

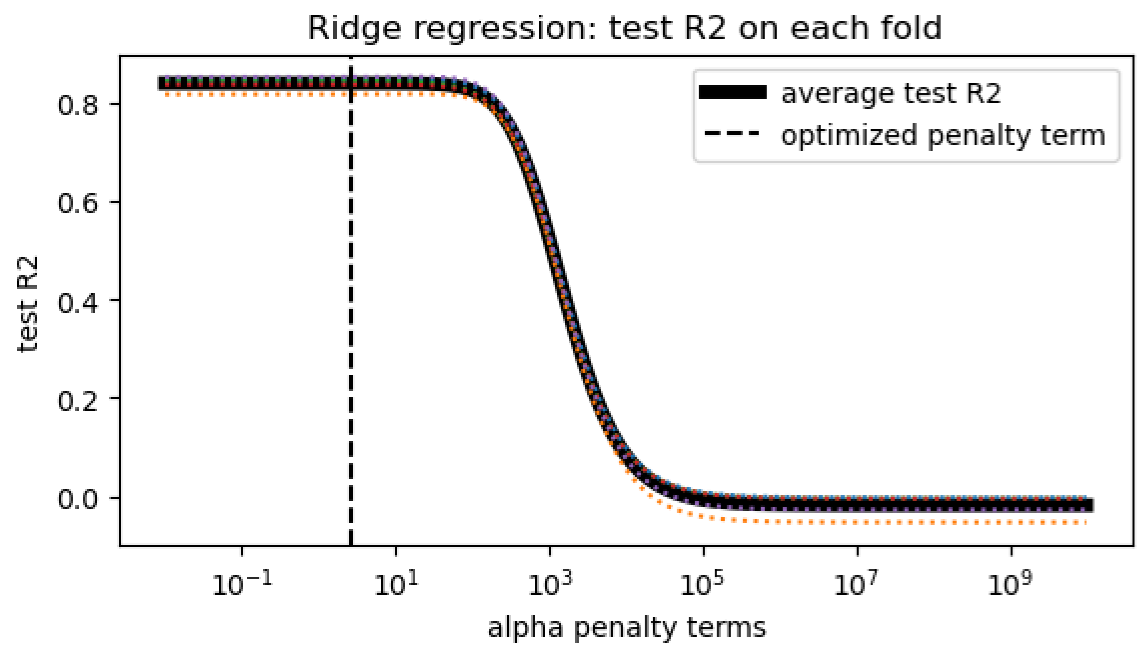
In [11]:
```
#model coefficients
ridge_coefs = ridge_results['coefs'].apply(pd.Series)
ridge_coefs.columns = ['X' + str(i) for i in range(1, 6)]
ridge_coefs.index = ridge_results['alpha']
ridge_coefs.reset_index(inplace = True)

#average the regression coefficients across each CV fold
ridge_coefs = ridge_coefs.groupby('alpha').mean()
ridge_coefs.reset_index(inplace = True)
```

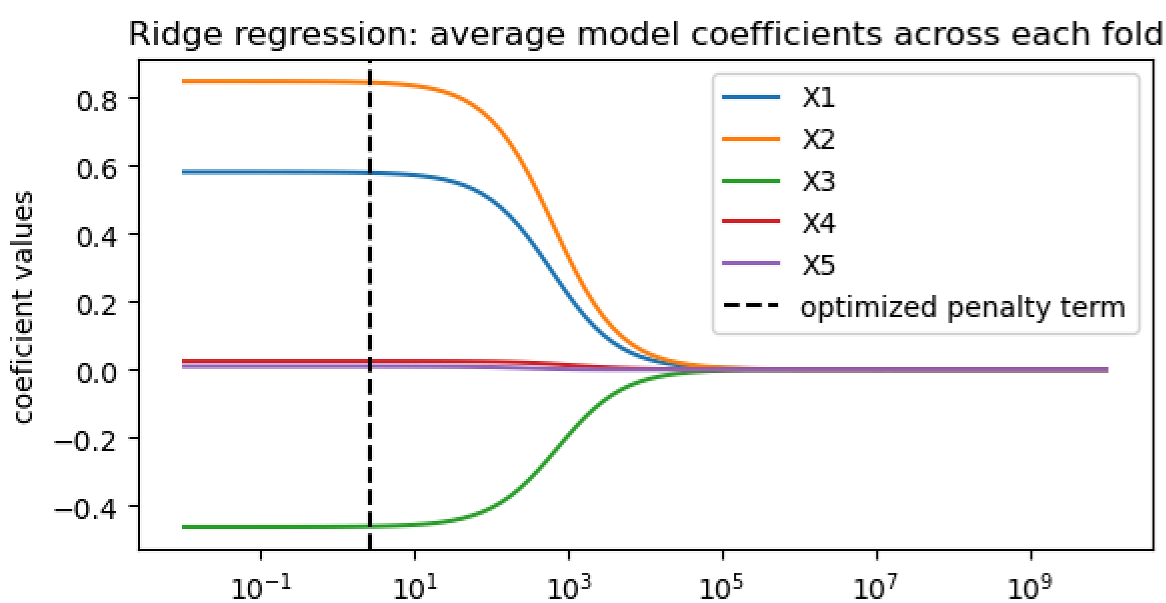
In [12]:
```
#filter each CV fold
ridge_cv1 = ridge_results[ridge_results['fold'] == 1]
ridge_cv2 = ridge_results[ridge_results['fold'] == 2]
ridge_cv3 = ridge_results[ridge_results['fold'] == 3]
ridge_cv4 = ridge_results[ridge_results['fold'] == 4]
ridge_cv5 = ridge_results[ridge_results['fold'] == 5]
```

In [13]:
```
#create subplots
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 3, ncols = 1, figsize = (6,10))

#plot the test MSE results
ax1.semilogx(ridge_avg['alpha'], ridge_avg['mse'], label = 'average test MSE', line
ax1.plot(ridge_cv1['alpha'], ridge_cv1['mse'], linestyle = 'dotted')
ax1.plot(ridge_cv2['alpha'], ridge_cv2['mse'], linestyle = 'dotted')
ax1.plot(ridge_cv3['alpha'], ridge_cv3['mse'], linestyle = 'dotted')
ax1.plot(ridge_cv4['alpha'], ridge_cv4['mse'], linestyle = 'dotted')
ax1.plot(ridge_cv5['alpha'], ridge_cv5['mse'], linestyle = 'dotted')
ax1.axvline(ridge_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax1.set_title('Ridge regression: test MSE on each fold')
ax1.set_xlabel('alpha penalty terms')
ax1.set_ylabel('test MSE')
ax1.legend()

#plot the R-squared results
ax2.semilogx(ridge_avg['alpha'], ridge_avg['r2'], label = 'average test R2', linewi
ax2.plot(ridge_cv1['alpha'], ridge_cv1['r2'], linestyle = 'dotted')
ax2.plot(ridge_cv2['alpha'], ridge_cv2['r2'], linestyle = 'dotted')
ax2.plot(ridge_cv3['alpha'], ridge_cv3['r2'], linestyle = 'dotted')
ax2.plot(ridge_cv4['alpha'], ridge_cv4['r2'], linestyle = 'dotted')
ax2.plot(ridge_cv5['alpha'], ridge_cv5['r2'], linestyle = 'dotted')
ax2.axvline(ridge_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax2.set_title('Ridge regression: test R2 on each fold')
ax2.set_xlabel('alpha penalty terms')
ax2.set_ylabel('test R2')
ax2.legend()

#plot the model coefficients
```

```
ax3.semilogx(ridge_coefs['alpha'], ridge_coefs['X1'], label = 'X1')
ax3.plot(ridge_coefs['alpha'], ridge_coefs['X2'], label = 'X2')
ax3.plot(ridge_coefs['alpha'], ridge_coefs['X3'], label = 'X3')
ax3.plot(ridge_coefs['alpha'], ridge_coefs['X4'], label = 'X4')
ax3.plot(ridge_coefs['alpha'], ridge_coefs['X5'], label = 'X5')
ax3.axvline(ridge_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax3.set_title('Ridge regression: average model coefficients across each fold')
ax3.set_xlabel('alpha penalty terms')
ax3.set_ylabel('coeficient values')
ax3.legend()

plt.tight_layout()
plt.show()
```


Ridge regression: test MSE on each fold
test MSE
alpha penalty terms
average test MSE
optimized penalty term


Ridge regression: test R2 on each fold
test R2
alpha penalty terms
average test R2
optimized penalty term


Ridge regression: average model coefficients across each fold
coeficient values
X1
X2
X3
X4
X5
optimized penalty term

alpha penalty terms

LASSO regression

Absolute value shrinkage penalty:

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

Lambda penalty optimization

Optimize the penalty term using k-fold cross-validation.

In [14]:
```
#concatenate the training data target and feature variables
lasso_df = pd.DataFrame(np.hstack((y_train.reshape(-1,1), X_train_scaled)))

#split the training data into k-folds
cv = 5
lasso_cv = k_folds_split(lasso_df, cv)
len(lasso_cv)
```

Out[14]: 5

In [15]:
```
#define the LASSO regression model
lasso = Lasso(fit_intercept = True)
lasso_results = []

for alpha in alphas:
    #set the penalty term
    lasso.set_params(alpha = alpha)

    #for each CV fold
    for i in range(cv):
        #define train and test sets
        X_train_cv = lasso_cv[i][1].drop(0, axis = 1)
        y_train_cv = lasso_cv[i][1][0]
        X_test_cv = lasso_cv[i][0].drop(0, axis = 1)
        y_test_cv = lasso_cv[i][0][0]

        #fit the LASSO regression model
        lasso.fit(X_train_cv, y_train_cv)

        #calculate test scores (test MSE and R-squared)
        mse = mean_squared_error(y_test_cv, lasso.predict(X_test_cv))
        r2 = r2_score(y_test_cv, lasso.predict(X_test_cv))

        #store the results
        lasso_results.append([alpha, i + 1, mse, r2, lasso.coef_])

lasso_results = pd.DataFrame(lasso_results)
```

```
lasso_results.columns = ['alpha','fold','mse','r2','coefs']
lasso_results.head()
```

Out[15]:

	alpha	fold	mse	r2	coefs
0	0.01	1	0.222246	0.851698	[0.5643975736821067, 0.8324467025096097, -0.46...
1	0.01	2	0.272661	0.835655	[0.5610502913182083, 0.8375397521097339, -0.44...
2	0.01	3	0.241162	0.833271	[0.5722743914893211, 0.8468691712765548, -0.45...
3	0.01	4	0.187040	0.860822	[0.573750473923607, 0.8408818519097174, -0.456...
4	0.01	5	0.292451	0.825955	[0.5860457381220797, 0.8319423282160593, -0.45...

Determine the optimal penalty term for the LASSO regression model.

In [16]:
```
#average the test MSE across each CV fold
lasso_avg = pd.DataFrame(lasso_results.drop('coefs', axis = 1).groupby('alpha').mea
lasso_avg.reset_index(inplace = True)

#determine the penalty term with the lowest test MSE
min_idx = lasso_avg['mse'].idxmin()
lasso_alpha = lasso_avg.at[min_idx, 'alpha']
print(f"LASSO optimal penalty term: {lasso_alpha}")
```

```
LASSO optimal penalty term: 0.01
```

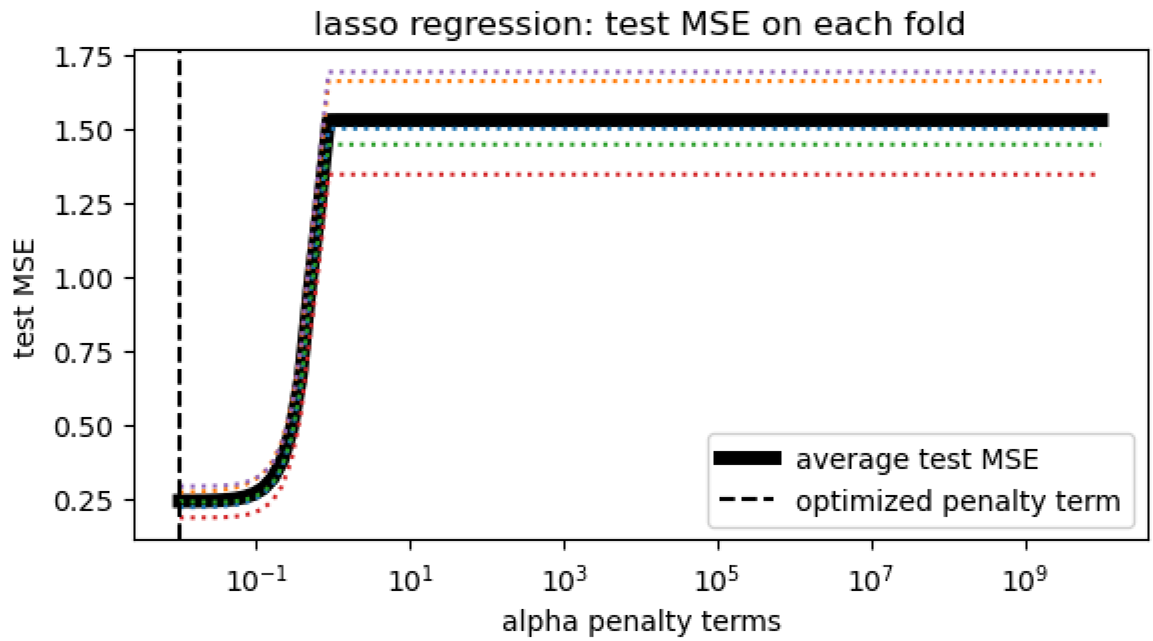
Visualize model performance

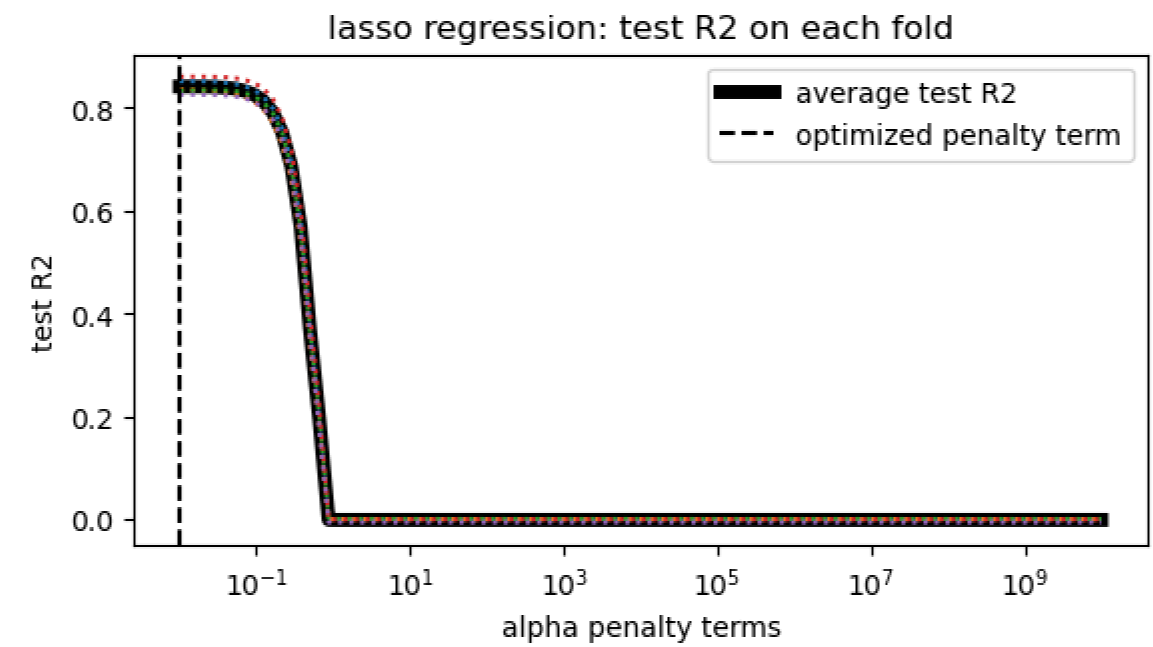
In [17]:
```
#model coefficients
lasso_coefs = lasso_results['coefs'].apply(pd.Series)
lasso_coefs.columns = ['X' + str(i) for i in range(1, 6)]
lasso_coefs.index = lasso_results['alpha']
lasso_coefs.reset_index(inplace = True)

#average the regression coefficients across each CV fold
lasso_coefs = lasso_coefs.groupby('alpha').mean()
lasso_coefs.reset_index(inplace = True)
```

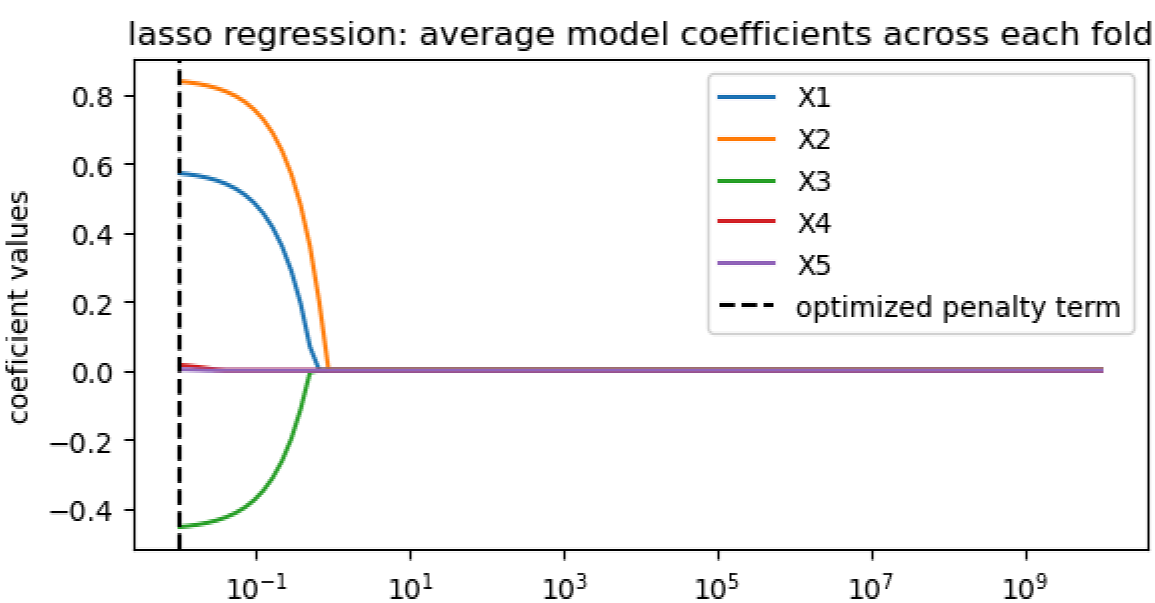
In [18]:
```
#filter each CV fold
lasso_cv1 = lasso_results[lasso_results['fold'] == 1]
lasso_cv2 = lasso_results[lasso_results['fold'] == 2]
lasso_cv3 = lasso_results[lasso_results['fold'] == 3]
lasso_cv4 = lasso_results[lasso_results['fold'] == 4]
lasso_cv5 = lasso_results[lasso_results['fold'] == 5]
```

In [19]:
```
#create subplots
fig, (ax1, ax2, ax3) = plt.subplots(nrows = 3, ncols = 1, figsize = (6,10))

#plot the test MSE results
ax1.semilogx(lasso_avg['alpha'], lasso_avg['mse'], label = 'average test MSE', line
ax1.plot(lasso_cv1['alpha'], lasso_cv1['mse'], linestyle = 'dotted')
ax1.plot(lasso_cv2['alpha'], lasso_cv2['mse'], linestyle = 'dotted')
ax1.plot(lasso_cv3['alpha'], lasso_cv3['mse'], linestyle = 'dotted')
ax1.plot(lasso_cv4['alpha'], lasso_cv4['mse'], linestyle = 'dotted')
```

```
ax1.plot(lasso_cv5['alpha'], lasso_cv5['mse'], linestyle = 'dotted')
ax1.axvline(lasso_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax1.set_title('lasso regression: test MSE on each fold')
ax1.set_xlabel('alpha penalty terms')
ax1.set_ylabel('test MSE')
ax1.legend()

#plot the R-squared results
ax2.semilogx(lasso_avg['alpha'], lasso_avg['r2'], label = 'average test R2', linewi
ax2.plot(lasso_cv1['alpha'], lasso_cv1['r2'], linestyle = 'dotted')
ax2.plot(lasso_cv2['alpha'], lasso_cv2['r2'], linestyle = 'dotted')
ax2.plot(lasso_cv3['alpha'], lasso_cv3['r2'], linestyle = 'dotted')
ax2.plot(lasso_cv4['alpha'], lasso_cv4['r2'], linestyle = 'dotted')
ax2.plot(lasso_cv5['alpha'], lasso_cv5['r2'], linestyle = 'dotted')
ax2.axvline(lasso_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax2.set_title('lasso regression: test R2 on each fold')
ax2.set_xlabel('alpha penalty terms')
ax2.set_ylabel('test R2')
ax2.legend()

#plot the model coefficients
ax3.semilogx(lasso_coefs['alpha'], lasso_coefs['X1'], label = 'X1')
ax3.plot(lasso_coefs['alpha'], lasso_coefs['X2'], label = 'X2')
ax3.plot(lasso_coefs['alpha'], lasso_coefs['X3'], label = 'X3')
ax3.plot(lasso_coefs['alpha'], lasso_coefs['X4'], label = 'X4')
ax3.plot(lasso_coefs['alpha'], lasso_coefs['X5'], label = 'X5')
ax3.axvline(lasso_alpha, linestyle = "--", color = 'black', label = "optimized pena
ax3.set_title('lasso regression: average model coefficients across each fold')
ax3.set_xlabel('alpha penalty terms')
ax3.set_ylabel('coeficient values')
ax3.legend()

plt.tight_layout()
plt.show()
```


lasso regression: test MSE on each fold
test MSE
alpha penalty terms
average test MSE
optimized penalty term


lasso regression: test R2 on each fold
test R2
alpha penalty terms
average test R2
optimized penalty term


lasso regression: average model coefficients across each fold
coeficient values
X1
X2
X3
X4
X5
optimized penalty term

alpha penalty terms

ElasticNet regression

Squared and absolute value shrinkage penalty:

$$RSS + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

$$\text{penalty} = \alpha \left(\text{l1\_ratio} \sum_{j=1}^{n} |\beta_j| + \frac{1 - \text{l1\_ratio}}{2} \sum_{j=1}^{n} \beta_j^2 \right)$$

l1_ratio = 0 emphasizes the λ_2 penalty and ElasticNet behaves like Ridge regression.

l1_ratio = 1 emphasizes the λ_1 penalty and ElasicNet behaves like LASSO regression.

Lambda penalty optimization

Optimize both penalty terms using k-fold cross-validation.

In [20]:
```
#define the grid space of hyper-parameters
param_grid = {
    'alpha': alphas,
    'l1_ratio': np.linspace(0.1, 0.9, 9)
}
```

In [21]:
```
#create the ElasticNet model
elastic_net = ElasticNet(fit_intercept = True)

#optimize the hyper-parameters using grid search
grid_search = GridSearchCV(estimator = elastic_net,
                           param_grid = param_grid,
                           cv = 5,
                           scoring = ['neg_mean_squared_error','r2'],
                           refit = 'neg_mean_squared_error',
                           n_jobs = -1)
grid_search.fit(X_train_scaled, y_train)
```

Out[21]:
```
▸ GridSearchCV
▸ estimator: ElasticNet
  ▸ ElasticNet
```

In [22]:
```
#display the optimized parameters for ElasticNet
grid_search.best_params_
```

Out[22]:
```
{'alpha': 0.01, 'l1_ratio': 0.9}
```

Given the *l1_ratio* is near 1, ElasticNet is behaving more similar to LASSO regression.

Model comparisons: Ridge, LASSO, and ElasticNet

Ridge regression

Optimized model coefficients

In [23]:
```
#Ridge model coefficients
ridge.set_params(alpha = ridge_alpha)
ridge.fit(X_train_scaled, y_train)
ridge_final_coefs = pd.Series(ridge.coef_, index = ['X' + str(i) for i in range(1,
ridge_final_coefs
```

Out[23]:
```
X1    0.579128
X2    0.845041
X3   -0.461721
X4    0.024354
X5    0.009486
dtype: float64
```

Model prediction accuracy

Test the model prediction accuracy.

In [24]:
```
#calculate the test MSE
ridge_mse = mean_squared_error(y_test, ridge.predict(X_test_scaled))
print(f"The test MSE of the Ridge regression model is {ridge_mse:.4f}.")
```

```
The test MSE of the Ridge regression model is 0.2771.
```

LASSO regression

Optimized model coefficients

In [25]:
```
#LASSO model coefficients
lasso.set_params(alpha = lasso_alpha)
lasso.fit(X_train_scaled, y_train)
lasso_final_coefs = pd.Series(lasso.coef_, index = ['X' + str(i) for i in range(1,
lasso_final_coefs
```

Out[25]:
```
X1    0.571167
X2    0.837952
X3   -0.453598
X4    0.014483
X5    0.000000
dtype: float64
```

Model prediction accuracy

In [26]:
```
#calculate the test MSE
lasso_mse = mean_squared_error(y_test, lasso.predict(X_test_scaled))
print(f"The test MSE of the LASSO regression model is {lasso_mse:.4f}.")
```

```
The test MSE of the LASSO regression model is 0.2763.
```

ElasticNet regression

Optimized model coefficients

In [27]:
```
#ElasticNet model coefficients
elastic_net = grid_search.best_estimator_
elastic_net_coefs = pd.Series(elastic_net.coef_, index = ['X' + str(i) for i in ran
elastic_net_coefs
```

Out[27]:
```
X1    0.571573
X2    0.838099
X3   -0.454131
X4    0.015460
X5    0.000214
dtype: float64
```

Model prediction accuracy

In [28]:
```
#calculate the test MSE
elastic_net_mse = mean_squared_error(y_test, elastic_net.predict(X_test_scaled))
print(f"The test MSE of the ElasticNet regression model is {elastic_net_mse:.4f}")
```

```
The test MSE of the ElasticNet regression model is 0.2764
```

Conclusion

All regularization methods including Ridge, LASSO, and ElasticNet regression resulted in similar model coefficients with almost equal test perfomance.