Sparse Regression

Ankur Ankan (s4753828), Kevin Jacobs(s4134621), Zhuoran Liu(s4594851)

codefile: lasso.py

1 Introduction

1.1 Lasso Regression using Coordinate Descent

Lasso Regression is a regularized linear regression technique in which we use L_1 regularization and get the following optimization problem:

$$\min_{\beta} \frac{1}{2} \sum_{\mu=1}^{p} \left(y^{\mu} - \sum_{i=1}^{n} \beta_i x_i^{\mu} \right)^2 \tag{1}$$

under the constraint that: $\sum_{i=1}^{n} \mid \beta_i \mid \leq t$. The iterative solution to this optimization problem using coordinate descent is given by:

$$\beta_j \leftarrow S\left(\frac{1}{p} \sum_{\mu} \widetilde{y}_j^{\mu} x_j^{\mu}, \gamma \right) \tag{2}$$

where:

$$\begin{aligned} \widetilde{y}_j^{\mu} &= y^{\mu} - \sum_{i \neq j} \beta_i x_i^{\mu} \\ S(\hat{\beta}, \gamma) &= sign(\hat{\beta})(\mid \hat{\beta} \mid - \gamma)_+ \end{aligned} \tag{3}$$

$S(\hat{\beta}, \gamma)$ can be written in simpler terms as:

$$S(\hat{\beta}, \gamma) = \begin{cases} \hat{\beta} - \gamma & \text{if } \hat{\beta} > 0 \text{and} \gamma < \mid \hat{\beta} \mid \\ \hat{\beta} + \gamma & \text{if } \hat{\beta} < 0 \text{and} \gamma < \mid \hat{\beta} \mid \\ 0 & \text{if } \gamma \geq \mid \hat{\beta} \mid \end{cases}$$

Lasso regression works particularly well in the cases when the number of features are huge. Unlike Ordinary Linear Regression, Lasso has the tendency to converge the weight values towards 0 and hence using only more important features and avoiding overfitting.

2 Sequential Gauss-Seidel rule for Lasso

Starting with the Lagrangian form of Lasso optimization problem, we need to minimize f given by:

$$f = -\frac{1}{2p}\sum_{\mu=1}^{p}\left(y^{\mu} - \sum_{i=1}^{n}\beta_i x_i^{\mu}\right)^2 + \sum_{i=1}^{n}\gamma \mid \beta \mid \tag{4}$$

Taking the derivative with respect to β and equating it to 0 we get:

$$\frac{\partial f}{\partial \beta_j} = \frac{1}{p}\sum_{\mu=1}^{p}\left(y^{\mu} - \sum_{i=1}^{n}\beta_i x_i^{\mu}\right) x_j^{\mu} + \gamma sign(\beta_j) = 0 \tag{5}$$

Putting it in matrix form we have:

$$-b_j + \sum_i \chi_{ij}\beta_i + \gamma sign(\beta_j) \tag{6}$$

$$\chi\beta' = b - \gamma sign(\beta) \tag{7}$$

Now from Gauss-Seidel rule we have the iterative solution of the equation $AX = b$ as:

$$x_i' = \frac{1}{A_{ii}}\left(b_i - \sum_{j>i} A_{ij}x_j - \sum_{j<i} A_{ij}x_j'\right) \tag{8}$$

Comparing the update rule with the equation we get:

$$\beta_i' = \frac{1}{\chi_{ii}}\left((b - \gamma sign(\beta))_i - \sum_{j>i}\chi_{ij}\beta_j - \sum_{j<i}\chi_{ij}\beta_j\right) \tag{9}$$

3 Research Questions

1. How does the accuracy of the model change for different values of γ ?
2. How does γ affect the absolute value of β ?
3. How does Lasso Regression perform when the features are correlated ?

4 Results

We are given a dataset of 50 samples each having 100 features. The first step is to standardize our data set using:

$$X_j = (X_j - mean(X_j))/std(X_j) \forall j = \{1, 2, ..., n_f eatures\} \tag{10}$$

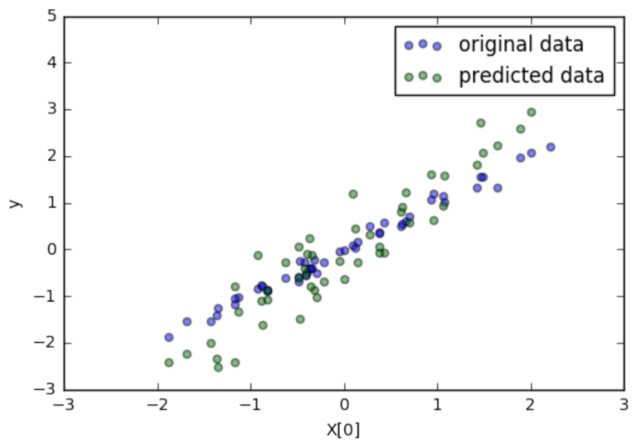

Figure 1: Original target variables with the predicted values against the 1st feature (X[0]) of the data. Model learned with $\gamma = 0.1$

We did a simple test for the fit of our learned parameter values shown in Fig. 1. We have plotted our predicted y against the most important feature i.e. the feature with the highest weight $X[0]$. We can see in the figure that even with a single feature we are able to predict the data quite well.

4.1 Variation of parameters with change in gamma

We start with random values for β sampled from a Normal Distribution and iteratively update each value of β one by one keeping the others fixed. In Fig. 2, we can see the variation of the weights with the iterations. We can see that the weights converge quite quickly in just 4 iterations and most of the weight values converge to 0 as we expect in the case of Lasso regression. Fig. 3 shows the number of parameters that converge to 0 for different values of γ. As we increase the value of γ, we penalize the weights more and hence more parameters start converging to 0 as we can see in the figure.

4.2 Variation of accuracy with change in gamma

In Fig. 4 we have plotted the mean squared error on the validation set for different values of γ. We see that when γ is really small we have a huge value of t and hence most of the parameters have non-zero weights and hence it results in overfitting because of which we are getting high mean squared error. Also in the case when γ is high we have a too constrained model resulting in underfitting and hence the high mean squared error. We get the best accuracy on the validation set for $\gamma = 0.22$.

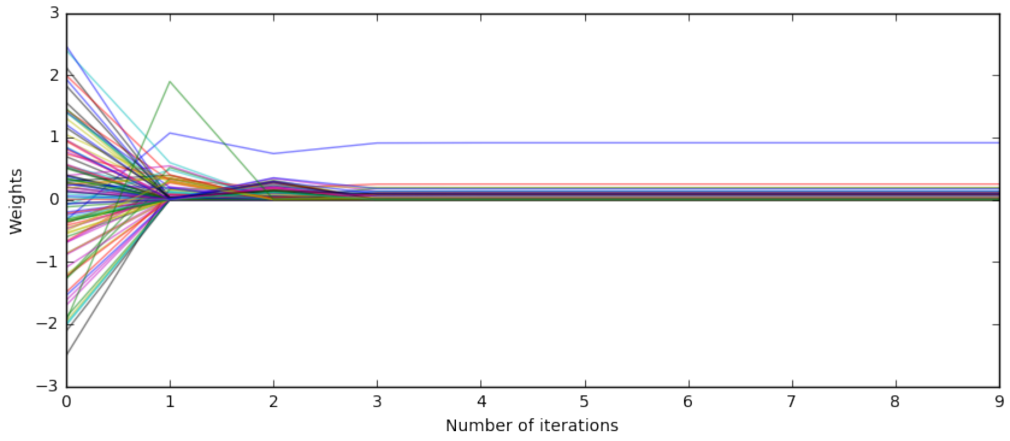

Figure 2: Change in weights β with iterations. In each iteration each β value is updated exactly once.

4.3 Performance in the correlated case

For the correlated case we have a dataset having 1000 samples with 3 features each. From the correlation matrix in Fig. 6, we can see that the 2rd feature is correlated with both 0 and 1. It is more correlated with 1st feature than the 0th one. Therefore for good predictions regression should assign highest weight to the 1st feature and least weight to 2nd feature. But we can see in Fig. 7 that both 0th and 2nd features are turned off for lower values of γ and then the 2nd feature has the highest weight, resulting in the poor performance of the mode.

We also compared the performance of the Lasso Regression with Ridge Regression in Fig. 5. We can see that Ridge regression performs better than Lasso for all the values of gamma.

5 Conclusion and Discussion

The LASSO (Least Absolute Shrinkage and Selection Operator) is a regression method that involves penalizing the absolute size of the regression coefficients. By penalizing we end up in a situation where some of the parameter estimates may be exactly zero. The larger the penalty applied, the further estimates are shrunk towards zero. This is convenient when we want some automatic feature selection algorithm or when our feature size is too huge.

But as we saw one of the major drawbacks of this method is that it performs poorly in the case of correlated features and arbitrarily assigns 0 weight to some of the correlated case resulting in very poor performance. In such cases ridge regression is a good alternative as it is biased towards assigning a non zero weight to all the features.

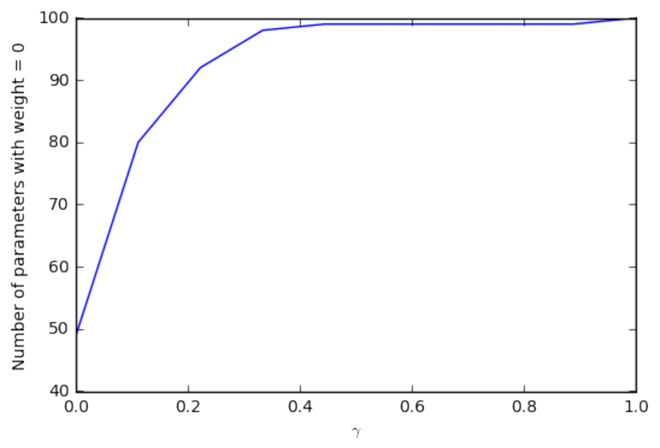

Figure 3: Number of parameters that have converged to 0 for various values of γ

6 Appendix

6.1 Code

```
import numpy as np
import pandas as pd

X_train = np.loadtxt('lasso_data/data1_input_train').T
y_train = np.loadtxt('lasso_data/data1_output_train')[:, np.newaxis]
X_val = np.loadtxt('lasso_data/data1_input_val').T
y_val = np.loadtxt('lasso_data/data1_output_val')[:, np.newaxis]

X_train = (X_train - X_train.mean(axis=0)) / (X_train.std(axis=0))
y_train = (y_train - y_train.mean(axis=0)) / (y_train.std(axis=0))
X_val = (X_val - X_val.mean(axis=0)) / (X_val.std(axis=0))
y_val = (y_val - y_val.mean(axis=0)) / (y_val.std(axis=0))


def S(beta, gamma):
    if gamma >= beta:
        return 0
    else:
        return beta - gamma if beta > 0 else beta + gamma

def train_lasso(X, y, gamma, all_betas=False, n_iter=100, init_weights=None):
    n_samples = X.shape[0]
    n_features = X.shape[1]
```

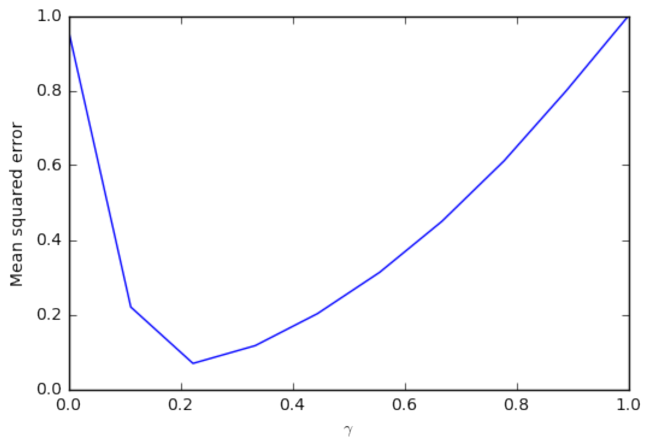

Figure 4: Mean Squared Error vs γ on validation dataset

```
    betas = np.zeros(shape=(n_iter + 1, n_features))
    if init_weights is not None:
        beta = init_weights
    else:
        beta = np.random.randn(n_features)
    betas[0, :] = beta

    for _ in range(n_iter):
        for j in range(n_features):
            # Calculate y_tilde
            y_tilde = np.zeros((n_samples,))
            for i in range(n_samples):
                for k in range(n_features):
                    y_tilde[i] = X[i, k] * beta[k]
            f1 = beta[j]
            for i in range(n_samples):
                f1 += X[i, j] * (y[i] - y_tilde[i])
            beta[j] = S(f1 / n_samples, gamma)
        betas[_ + 1, :] = beta
    if all_betas:
        return betas
    else:
        return beta

import matplotlib.pyplot as plt

betas = train_lasso(X_train, y_train, gamma=0.1, all_betas=True)
fig = plt.figure(figsize=(10, 4))
```

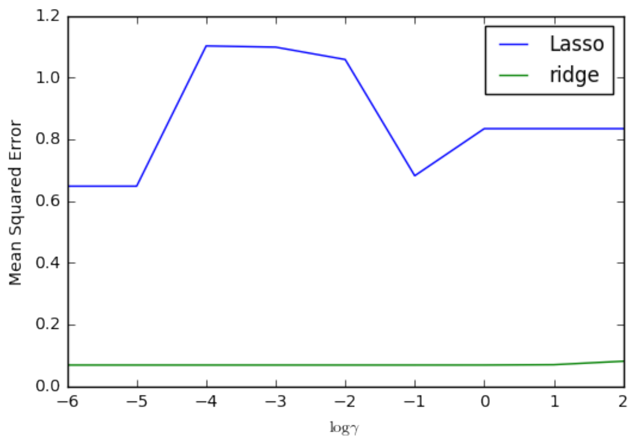

Figure 5: Comparison of Lasso and Ridge regression when the features are correlated

```
for feature in range(100):
    plt.plot(np.arange(10), betas[:10, feature], alpha=0.5)
    plt.scatter(np.arange(10), betas[:10, feature], alpha=0.3)

plt.ylabel('Weights')
plt.xlabel('Number of iterations')
plt.show()


##############################################
######### Correlated Case  ###################
##############################################
import numpy as np
import pandas as pd
from sklearn.cross_validation import train_test_split

X = np.loadtxt('lasso_data/X_cor', delimiter=',').T
y = np.loadtxt('lasso_data/y_cor', delimiter=',')[:, np.newaxis]
X = (X - X.mean(axis=0)) / (X.std(axis=0))
y = (y - y.mean(axis=0)) / (y.std(axis=0))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

import matplotlib.pyplot as plt

temp = np.array(betas)
for i in range(3):
```

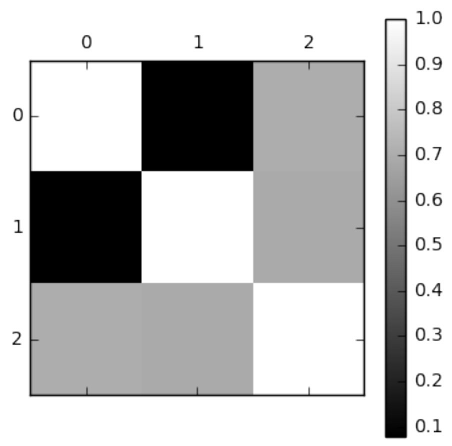

Figure 6: The correlation matrix between the 3 features of the given dataset

```
    plt.plot(np.linspace(0, 1, 10), temp[:, i], label=str(i))

plt.legend()
plt.xlabel("$ \gamma $")
plt.ylabel("Weights")
plt.show()


import pandas as pd
df_X = pd.DataFrame(X_train)
df_X.corr()

plt.matshow(df_X.corr(), cmap=plt.cm.gray)
plt.colorbar()

from sklearn.linear_model import LinearRegression

l_errors = []
r_errors = []
o_errors = []
gammas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2]
for g in gammas:
    beta_lasso = train_lasso(X_train, y_train, gamma=g)
    l_errors.append(mean_squared_error(y_test, predict(X_test, beta=beta_lasso)))

    clf = Ridge(alpha=g)
    clf.fit(X_train, y_train)
    r_errors.append(mean_squared_error(y_test, clf.predict(X_test)))
```

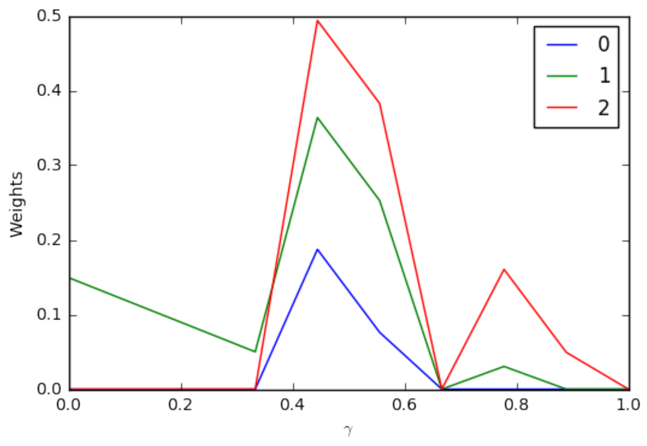

Figure 7: Converged values of β for different values of γ in the correlated dataset

```
    clf = LinearRegression()
    clf.fit(X_train, y_train)
    o_errors.append(mean_squared_error(y_test, clf.predict(X_test)))


plt.plot(range(-6, 3), l_errors, label='Lasso')
plt.plot(range(-6, 3), r_errors, label='ridge')
plt.xlabel('$ \log \gamma $ ')
plt.ylabel('Mean Squared Error')
plt.legend()
```