Regularised Linear Models-Shizhi Chen-10307389

Assignment 1: Add your code on calculating the Mean Squared Error (MSE) to the provided code.

Note that MSE = RSS/n, where n is the number of instances in a test data set.

Python Code:

```python
# Load the libraries needed
%matplotlib inline
import csv
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge, Lasso
import time

# Seed random number generator
np.random.seed(123)

# Helper functions for linear models
def linear ( X, coefficients ):
    return np.dot(X, np.transpose(coefficients), )
```

```python
def fit_linear( X, y, features=None ):
    """
        Returns the coefficients of a linear model fit to X,y.
        If features is a list of integers, then fit will ignore
        any features whose index is not in the list.
        ( Returned coefficients for these features will be set
        to 0. )
    """
    if features is not None:
        # Make a mask
        tot_num_features = np.shape(X)[-1]
        mask = np.zeros((1,tot_num_features))
        mask[0,features] = 1.
        # Zero out all irrellevant features
        X = X * mask

    # Do linear least squares fit
    clf = LinearRegression(fit_intercept=False)
    clf.fit(X, y)
    return clf.coef_


def ridge_regression( X, y, lam=1.0, features=None ):
    """
        Identical to fit_linear, but performs ridge regression
        with weight penalty alpha (alternatively known as lambda)
    """
    if features is not None:
        # Make a mask
        tot_num_features = np.shape(X)[-1]
        mask = np.zeros((1,tot_num_features))
        mask[0,features] = 1.
        # Zero out all irrellevant features
        X = X * mask

    # Do ridge regression fit
    clf = Ridge(alpha=lam, fit_intercept=False)
    clf.fit(X, y)
    return clf.coef_
```

```
def lasso_regression( X, y, lam=1.0, features=None ):
    """
        Identical to fit_linear, but performs lasso regression
        with weight penalty alpha (alternatively known as lambda)
    """
    if features is not None:
        # Make a mask
        tot_num_features = np.shape(X)[-1]
        mask = np.zeros((1,tot_num_features))
        mask[0,features] = 1.
        # Zero out all irrellevant features
        X = X * mask

    # Do ridge regression fit
    clf = Lasso(alpha=lam, fit_intercept=False, max_iter=1e5)
    clf.fit(X, y)
    return clf.coef_
```

```
# MSE score
# You should modify the function mse_loss to compute
# the Mean Square Error given a dataset X, y, and learned
# linear model with coefficients learned_coeff.
def mse_loss( X, y, learned_coefficients ):
    y_pred = linear(X, learned_coefficients)
    rss=np.sum(np.square(y_pred - y))
    return rss/len(X)
```

• Run the provided code (where the hyper-parameter λ is given) to train 3 different linear models on the data set, class10_training_a, with the OLS, ridge regression and LASSO learning algorithms, respectively. Then, run your modified code to calculate the MSE on the test set, class10_test, with 3 trained linear models, respectively. [2 marks]

```
# Load generated data
X, [y] = np.load("./class10_training_a.npy")
X_test, [y_test] = np.load("./class10_test.npy")

lin_reg_train_loss = mse_loss(X, y, fit_linear(X, y))
ridge_train_loss = mse_loss(X, y, ridge_regression(X, y, lam=1.0))
lasso_train_loss = mse_loss(X, y, lasso_regression(X, y, lam=0.003))

print("Train Loss ({:d} datapoints)".format(len(X)))
print("Linear Regression:     {:.3f}".format(lin_reg_train_loss))
print("Ridge Regression:      {:.3f}".format(ridge_train_loss))
print("LASSO:                 {:.3f}".format(lasso_train_loss))

print("\nTest Loss ({:d} datapoints)".format(len(X)))
# Include your code for calculating the loss on the test set (X_test, y_test)
# (And remember NEVER to train on the test set)

lin_reg_test_loss = mse_loss(X_test, y_test, fit_linear(X, y))
ridge_test_loss = mse_loss(X_test, y_test, ridge_regression(X, y, lam=1.0))
lasso_test_loss = mse_loss(X_test, y_test, lasso_regression(X, y, lam=0.003))

print("Linear Regression:     {:.3f}".format(lin_reg_test_loss))
print("Ridge Regression:      {:.3f}".format(ridge_test_loss))
print("LASSO:                 {:.3f}".format(lasso_test_loss))
```

```
Train Loss (100 datapoints)
Linear Regression:     0.057
Ridge Regression:      0.057
LASSO:                 0.058

Test Loss (100 datapoints)
Linear Regression:     0.169
Ridge Regression:      0.167
LASSO:                 0.158
```

From the results, we can see that when the number of examples larger than the number of features, these three models have similar MSE errors on the training dataset. However, in the testing dataset, OLS regression may suffer overfitting problem so it has the highest MSE errors. In addition, LASSO can interpret the models better than Ridge due to the LASSO penalty term has the effect of forcing some insignificant coefficients to zero which cannot achieve in the Ridge.

• Run the provided code (where the hyper-parameter λ is given) to train 2 regularised linear models on the data set, class10_training_b, with ridge regression and LASSO learning algorithms, respectively. Then, run your modified code to calculate the MSE on the test set, class10_test, with 2 trained regularised linear regression models, respectively. Comment on why the OLS cannot be applied to this training data set. [3 marks]

```
# Load generated data
X, [y] = np.load("./class10_training_b.npy")

ridge_train_loss = mse_loss(X, y, ridge_regression(X, y, lam=10))
lasso_train_loss = mse_loss(X, y, lasso_regression(X, y, lam=0.1))

print("Train Loss ({:d} datapoints)".format(len(X)))
# Why do we not use Linear Regression?
print("Ridge Regression:      {:.3f}".format(ridge_train_loss))
print("LASSO:                 {:.3f}".format(lasso_train_loss))

print("\nTest Loss ({:d} datapoints)".format(len(X)))
# Include your code for calculating the loss on the test set (X_test, y_test)
ridge_test_loss = mse_loss(X_test, y_test, ridge_regression(X, y, lam=10))
lasso_test_loss = mse_loss(X_test, y_test, lasso_regression(X, y, lam=0.1))

print("Ridge Regression:      {:.3f}".format(ridge_test_loss))
print("LASSO:                 {:.3f}".format(lasso_test_loss))
```

```
Train Loss (25 datapoints)
Ridge Regression:     0.047
LASSO:                0.146

Test Loss (25 datapoints)
Ridge Regression:     0.715
LASSO:                0.655
```

We calculate the optimal parameters in OSL by $\hat{\boldsymbol{\beta}} = \boxed{(X^T X)}^{-1} X^T \boldsymbol{y}$. In this training data set, the number of training examples is smaller than that of features which means that the XTX will be singular and as such the matrix cannot be invertible. Therefore, OLS cannot be applied to this training data set.

Assignment 2: Run the provided code to train LASSO on the training subset, class10_auto_train, with λ = 0.05, 0.5, respectively. Calculate the MSE on the test subset, class10_auto_test, with those trained LASSO models with 2 different λ values, respectively. Based on your observation, comment on the non-zero features achieved by LASSO when different λ values are used. [2marks]

```
# Automatic feature selection with LASSO

# Load data
X_train, [y_train] = np.load("./class10_auto_train.npy")
X_test, [y_test] = np.load("./class10_auto_test.npy")

lam = 0.05
learned_features = lasso_regression(X_train, y_train, lam=lam)
nonzero_features = np.argwhere(~ np.isclose(learned_features, 0.)).squeeze()

print("Fitted LASSO with lambda {:.3f}, learned parameters:".format(lam))
print(learned_features)
print("Non-zero features : {} ({:d} total)".format(list(nonzero_features), len(nonzero_features)))

train_loss = mse_loss(X_train, y_train, learned_features)
print("Train error: {:.3f}".format(train_loss))

test_loss = mse_loss(X_test, y_test, learned_features)
print("Test error:  {:.3f}".format(test_loss))
```

```
Fitted LASSO with lambda 0.050, learned parameters:
[ 0.37465549 -0.5768899   0.          0.07039306 -0.          0.
  0.          0.         -0.          0.          0.         -0.
 -0.02611483 -0.          0.        ]
Non-zero features : [0, 1, 3, 12] (4 total)
Train error: 0.464
Test error:  0.517
```

Change the λ = 0.5,

```
Fitted LASSO with lambda 0.500, learned parameters:
[ 0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.  -0.   0.   0.   0.  -0.]
Non-zero features : [] (0 total)
Train error: 0.949
Test error:  1.101
```

From the results, we can see that when λ = 0.5, there is no non-zero feature. But the model with λ = 0.05 has four non-zero features. This is because when the parameter λ becomes larger, the penalty for features' coefficients is heavier and the penalty term has the effect of forcing some of the coefficients to zero.

Assignment 3: Apply a proper method learnt in Lectures 8-10 to find out the optimal λ value used in the LASSO learning on the Automobile data set from λ = 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, and 1.0. Comment on the non-zero features obtained by the LASSO trained with the optimal λ value by comparing them to those achieved in Class 9. It is essential to justify why the method you use is appropriate to find out the optimal λ value. [3 marks]

Code:

```
# Load data
data = []
continuous_features = [ 0, 1, 9, 10, 11, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 25 ]

# Original data is from https://archive.ics.uci.edu/ml/datasets/automobile
with open('./automobile.csv', 'r') as csvfile:
    csvreader = csv.reader(csvfile, delimiter=',', quotechar='\"')
    for row in csvreader:
        try:
            # Get all continuous rows
            data.append([float(row[i]) for i in continuous_features])
        except:
            continue # skip this row since data-processing failed

data = np.array(data)
y = data[:, 0] # target is first value
X = data[:, 1:] # training data is the rest

# Normalize the data to zero mean and unit std
X = (X - np.mean(X, axis=0, keepdims=True)) / np.std(X, axis=0, keepdims=True)
y = (y - np.mean(y)) / np.std(y)

print("Sucessfully loaded {:d} entries.\n".format(len(X)))

# Implement your method for selecting an appropriate lambda below:
# K-fold Cross validation
# Helper function for getting folds
def get_fold(X, y, fold, num_folds):
    folds_X = np.array_split(X, num_folds)
    test_X = folds_X.pop(fold)
    train_X = np.concatenate(folds_X)
    folds_y = np.array_split(y, num_folds)
    test_y = folds_y.pop(fold)
    train_y = np.concatenate(folds_y)
    return train_X, train_y, test_X, test_y
```

```
lam_value=[0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0]
num_folds=10
lasso_loss=[]

for lam in lam_value:

    print( "Running LASSO model with lambda {:.3f} in the {:d}-folds cross validation:".format(lam,num_folds))

    losses=[]

    for i in range(num_folds):
        #Get data for fold
        train_X, train_y, test_X, test_y = get_fold(X, y, i, num_folds)

        #Calculate lasso's loss
        lasso_loss = mse_loss(X_test, y_test, lasso_regression(train_X, train_y, lam))
        losses.append(lasso_loss)
        print("Lasso loss for fold {:d}/{:d}: {:.5f}".format(i+1, num_folds, lasso_loss))

    learned_features = lasso_regression(X_train, y_train, lam=lam)
    nonzero_features = np.argwhere(~ np.isclose(learned_features, 0.)).squeeze()
    print("Non-zero features : {} ({:d} total)".format(list(nonzero_features), len(nonzero_features)))

    lasso_mean_loss=np.mean(losses)
    print("The mean loss of the {:d}-fold cross validation is: {:.5f}\n ".format(num_folds,lasso_mean_loss))
```

Output:

```
Sucessfully loaded 160 entries.

Running LASSO model with lambda 0.001 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.39382
Lasso loss for fold 2/10: 0.38452
Lasso loss for fold 3/10: 0.38637
Lasso loss for fold 4/10: 0.39711
Lasso loss for fold 5/10: 0.38335
Lasso loss for fold 6/10: 0.35406
Lasso loss for fold 7/10: 0.40312
Lasso loss for fold 8/10: 0.45759
Lasso loss for fold 9/10: 0.41485
Lasso loss for fold 10/10: 0.39605
Non-zero features : [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] (14 total)
The mean loss of the 10-fold cross validation is: 0.39708

Running LASSO model with lambda 0.003 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.39585
Lasso loss for fold 2/10: 0.38701
Lasso loss for fold 3/10: 0.38814
Lasso loss for fold 4/10: 0.39666
Lasso loss for fold 5/10: 0.38494
Lasso loss for fold 6/10: 0.35708
Lasso loss for fold 7/10: 0.40226
Lasso loss for fold 8/10: 0.45432
Lasso loss for fold 9/10: 0.41389
Lasso loss for fold 10/10: 0.39803
Non-zero features : [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] (14 total)
The mean loss of the 10-fold cross validation is: 0.39782

Running LASSO model with lambda 0.010 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.41442
Lasso loss for fold 2/10: 0.40435
Lasso loss for fold 3/10: 0.40583
Lasso loss for fold 4/10: 0.40178
Lasso loss for fold 5/10: 0.39723
Lasso loss for fold 6/10: 0.37217
Lasso loss for fold 7/10: 0.40894
Lasso loss for fold 8/10: 0.45269
Lasso loss for fold 9/10: 0.41884
Lasso loss for fold 10/10: 0.41254
Non-zero features : [0, 1, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13] (12 total)
The mean loss of the 10-fold cross validation is: 0.40888

Running LASSO model with lambda 0.030 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.43564
Lasso loss for fold 2/10: 0.43398
Lasso loss for fold 3/10: 0.44052
Lasso loss for fold 4/10: 0.43821
Lasso loss for fold 5/10: 0.42600
Lasso loss for fold 6/10: 0.41045
Lasso loss for fold 7/10: 0.43287
Lasso loss for fold 8/10: 0.46014
Lasso loss for fold 9/10: 0.45351
Lasso loss for fold 10/10: 0.43879
Non-zero features : [0, 1, 3, 9, 12] (5 total)
The mean loss of the 10-fold cross validation is: 0.43701

Running LASSO model with lambda 0.100 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.51963
Lasso loss for fold 2/10: 0.53472
Lasso loss for fold 3/10: 0.53559
Lasso loss for fold 4/10: 0.53555
Lasso loss for fold 5/10: 0.52501
Lasso loss for fold 6/10: 0.51972
Lasso loss for fold 7/10: 0.52888
Lasso loss for fold 8/10: 0.54930
Lasso loss for fold 9/10: 0.52022
Lasso loss for fold 10/10: 0.52790
Non-zero features : [0, 1] (2 total)
The mean loss of the 10-fold cross validation is: 0.52965
```

```
Running LASSO model with lambda 0.300 in the 10-folds cross validation:
Lasso loss for fold 1/10: 0.65777
Lasso loss for fold 2/10: 0.74747
Lasso loss for fold 3/10: 0.71072
Lasso loss for fold 4/10: 0.70138
Lasso loss for fold 5/10: 0.68053
Lasso loss for fold 6/10: 0.68250
Lasso loss for fold 7/10: 0.71621
Lasso loss for fold 8/10: 0.73880
Lasso loss for fold 9/10: 0.69209
Lasso loss for fold 10/10: 0.68397
Non-zero features : [0, 1] (2 total)
The mean loss of the 10-fold cross validation is: 0.70114

Running LASSO model with lambda 1.000 in the 10-folds cross validation:
Lasso loss for fold 1/10: 1.10097
Lasso loss for fold 2/10: 1.10097
Lasso loss for fold 3/10: 1.10097
Lasso loss for fold 4/10: 1.10097
Lasso loss for fold 5/10: 1.10097
Lasso loss for fold 6/10: 1.10097
Lasso loss for fold 7/10: 1.10097
Lasso loss for fold 8/10: 1.10097
Lasso loss for fold 9/10: 1.10097
Lasso loss for fold 10/10: 1.10097
Non-zero features : [] (0 total)
The mean loss of the 10-fold cross validation is: 1.10097
```

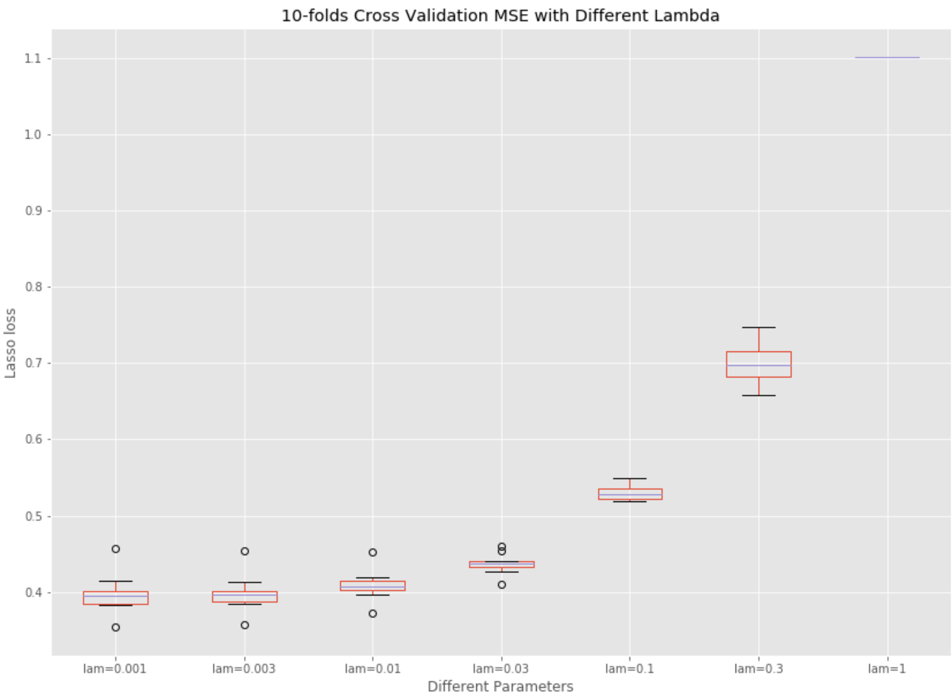
Visualize the results:

```python
import pandas as pd
plt.style.use("ggplot")

df=pd.DataFrame()
df["lam=0.001"]=losses1
df["lam=0.003"]=losses2
df["lam=0.01"]=losses3
df["lam=0.03"]=losses4
df["lam=0.1"]=losses5
df["lam=0.3"]=losses6
df["lam=1"]=losses7
plt.figure(figsize=(14,10))

df.boxplot()
plt.title("10-folds Cross Validation MSE with Different Lambda")
plt.xlabel("Different Parameters")
plt.ylabel("Lasso loss")

plt.show()
```



K-fold cross-validation is an appropriate method for finding the best λ in LASSO. This is because in K fold cross-validation, data can always be used in training and verification in order, so feature selection will be more objective.

From the result and the boxplots, we can see that the model with λ=0.001 have the best performance which have the lowest mean MSE=0.39708. The Non-zero features in this model are [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14] (14 total).

Forwards stepwise features selection:

```
Sucessfully loaded 160 entries.

Performing forwards stepwise feature selection using MSE as criteria...
Round 1, selected feature 1 with score 0.729
Round 2, selected feature 0 with score 0.493
Round 3, selected feature 3 with score 0.440
Round 4, selected feature 2 with score 0.435
Round 5, selected feature 9 with score 0.431
Round 6, selected feature 12 with score 0.429
Round 7, selected feature 13 with score 0.417
Round 8, selected feature 4 with score 0.416
Round 9, selected feature 10 with score 0.414
Round 10, selected feature 5 with score 0.412
Round 11, selected feature 11 with score 0.410
Round 12, selected feature 6 with score 0.409
Round 13, selected feature 7 with score 0.409
Round 14, selected feature 14 with score 0.408
Round 15, selected feature 8 with score 0.408
Best features were [1, 0, 3, 2, 9, 12, 13, 4, 10, 5, 11, 6, 7, 14, 8] (15 total) with score 0.408
```

Compared with forwards stepwise features selection we used in Class 9, we can see that the regularized linear model achieves the similar result with 15 features and MSE=0.408.