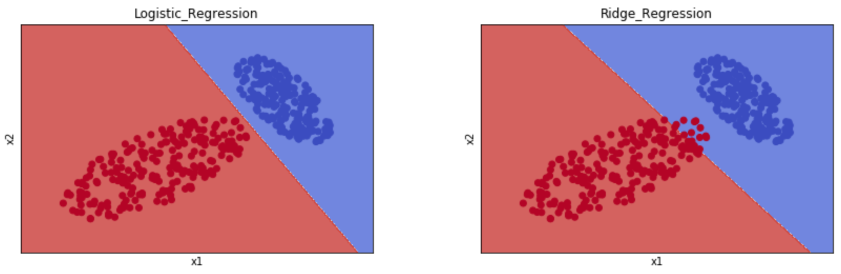
Machine Learning & Data Mining
Caltech CS/CNS/EE 155
Homework 2

Changhao Xu
UID: 2103530
January 20th, 2020

1 Comparing Different Loss Functions

Question A: Squared loss is to minimize the distance between data points and model boundary. In this case, points that are far away from the classification boundary are heavily penalized regardless of whether they are misclassified or not.

Question B: Logistic regression behaves better than ridge regression here. Logistic regression uses log loss while ridge regression uses squared loss; as explained in Question A, squared loss are heavily affected by the points far away from the classification boundary and thus cannot classify correctly.



Question C:

$$\nabla_w L_{hinge} = \begin{cases} 0 & y\mathbf{w}^T\mathbf{x} \geq 1 \\ -y\mathbf{x} & y\mathbf{w}^T\mathbf{x} < 1 \end{cases}$$

$$\nabla_w L_{log} = \frac{-y\mathbf{x}}{e^{y\mathbf{w}^T\mathbf{x}} + 1}$$

(1/2, 3): $\nabla_w L_{hinge}$ = (-1, -1/2, -3), $\nabla_w L_{log}$ = (-0.37754, -0.18877, -1.13262)
(2, -2): $\nabla_w L_{hinge}$ = 0, $\nabla_w L_{log}$ = (-0.11920, -0.23841, 0.23841)
(-3, 1): $\nabla_w L_{hinge}$ = 0, $\nabla_w L_{log}$ = (0.04743, -0.14228, 0.04743)

Question D: Hinge loss gradient will converge to 0 when the point is correctly classified ($y\mathbf{w}^T\mathbf{x} \geq 1$); while log loss gradient will not converge to 0 unless **x** = 0. For a linearly separable dataset, suppose a hyperplane with weight **w** that classify all points correctly. In order to eliminate training error, simply scale the weight vector **w** to a sufficiently large vector, so that $y\mathbf{w}^T\mathbf{x} \geq 1$ for all points **x** in the dataset.

Question E: If learning objective of SVM is to minimize just L_{hinge}, then any hyperplane with a properly scaled weight vector **w** can eliminate training error and result in 0 gradient. Therefore, in order for an SVM to be a "maximum margin" classifier, an additional penalty term $\lambda\|w\|^2$ must be added, where $\|w\|^2$ addresses the margin between the training points and the boundary.

Machine Learning & Data Mining
Caltech CS/CNS/EE 155
Homework 2

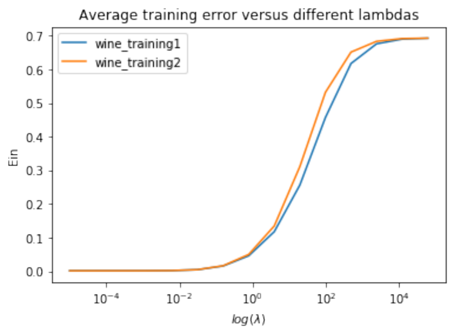
Changhao Xu
UID: 2103530
January 20th, 2020

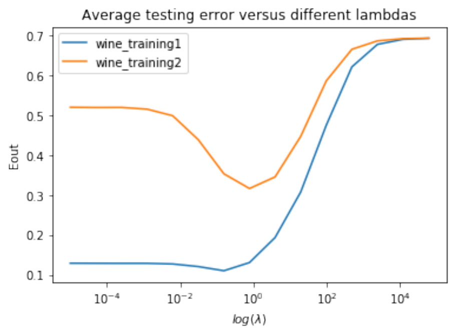
2 Effects of Regularization

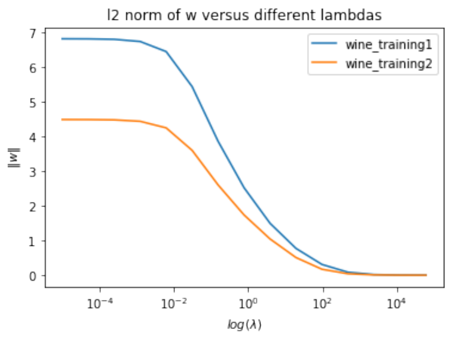
Question A: Adding the penalty term will increase the training (in-sample) error, because training without regularization will minimize training error, while regularization term will limit this fitting performance. Adding a penalty term cannot always decrease the out-of-sample errors (this only happens when overfitting occurs). When the model is simple and underfitting occurs, adding regularization will increase out-of-sample error.

Question B: l_0 regularization is not continuous, and thus is hard to use methods like stochastic gradient descent to optimize.

Question C:







Question D: Training error with training1 is overall smaller than that of training2. Initially with little regularization, both have overfitting ($E_{in} = 0$), then with better regularization, training1 have more data points and thus perform better. Testing error with training1 is overall smaller than that of training2, and both experience a decrease and then increase of E_{out}. Initially with little regularization, both have overfittings (E_{out} is big compared with E_{in}), then with better regularization, both model have a decrease in E_{out}, while training1 have more data points and thus perform better. As λ becomes too big, both model suffer from underfitting, and have an increase in E_{out}.

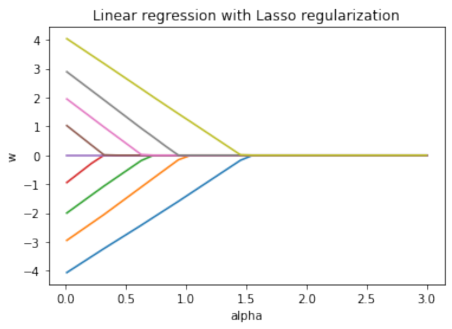
Question E: When λ increase, training error always increase due to adding of penalty term. As for testing error, initially testing error is big due to overfitting, then with regularization the model performs better, so testing error decrease, but with too large λ the model suffer from underfitting, so testing error increase.

Question F: When λ increase, l_2 norm of $\mathbf{w}$ decrease because w have fewer coefficients as model becomes simpler.

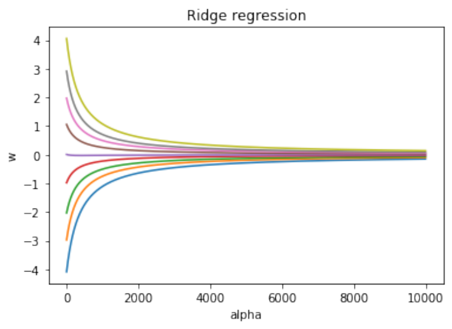
Question G: I would choose λ = 0.78125, which has smallest testing error.

3 Lasso (l_1) vs. Ridge (l_2) Regularization

Question A: i.



ii.



iii. As alpha increases, the weights in Lasso regression decrease linearly until they are close to 0, and then they reach exactly 0, so number of model weights that are exactly zero increases nearly linearly with the increase of alpha. Meanwhile, in Ridge regression, as alpha increases, the weights decrease exponentially

Changhao Xu
UID: 2103530

but the weights doesn't reach exactly 0.

Question B: i.

$$\begin{aligned}\nabla_w\Big[\|\mathbf{y}-\mathbf{x}w\|^2+\lambda\|w\|_1\Big] &= \nabla_w\Big[(\mathbf{y}^T-\mathbf{x}^Tw)(\mathbf{y}-\mathbf{x}w)+\lambda\|w\|_1\Big]\\ &= \nabla_w\Big[\mathbf{y}^T\mathbf{y}-\mathbf{y}^T\mathbf{x}w-\mathbf{x}^T\mathbf{y}w+\mathbf{x}^T\mathbf{x}w^2+\lambda\|w\|_1\Big]\\ &= -\mathbf{y}^T\mathbf{x}-\mathbf{x}^T\mathbf{y}+2w\mathbf{x}^T\mathbf{x}+\lambda\nabla_w\|w\|_1\\ &= 0\end{aligned}$$

$$w=\frac{1}{2}(\mathbf{x}^T\mathbf{x})^{-1}\Big[\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}-\lambda\nabla_w\|w\|_1\Big], where \nabla_w\|w\|_1=\begin{cases}sgn(w) & w\neq 0\\ [-1,+1] & w=0\end{cases}$$

In this case, when w = 0, $w=\frac{1}{2}(\mathbf{x}^T\mathbf{x})^{-1}\Big[\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}-\lambda[-1,+1]\Big]=0$, which requires that $\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}-\lambda[-1,+1]=0$, i.e. $\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}\leq\lambda$. Therefore,

$$w=\begin{cases}\frac{1}{2}(\mathbf{x}^T\mathbf{x})^{-1}\Big[\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}-\lambda sgn(w)\Big] & \mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}>\lambda\\ 0 & \mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}\leq\lambda\end{cases}$$

ii. When $\lambda=0$, $\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}>\lambda$ and w = $\frac{1}{2}(\mathbf{x}^T\mathbf{x})^{-1}\Big[\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}\Big]\neq 0$.
Therefore, when w = 0, $\lambda_{min}=\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}$

iii. When w becomes **w** and Lasso becomes Ridge,

$$\begin{aligned}\nabla_{\mathbf{w}}\Big[\|\mathbf{y}-\mathbf{x}\mathbf{w}\|^2+\lambda\|\mathbf{w}\|_2^2\Big] &= \nabla_{\mathbf{w}}\Big[(\mathbf{y}^T-\mathbf{w}^T\mathbf{x}^T)(\mathbf{y}-\mathbf{x}\mathbf{w})+\lambda\mathbf{w}^T\mathbf{w}\Big]\\ &= \nabla_{\mathbf{w}}\Big[\mathbf{y}^T\mathbf{y}-\mathbf{y}^T\mathbf{x}\mathbf{w}-\mathbf{w}^T\mathbf{x}^T\mathbf{y}+\mathbf{w}^T\mathbf{x}^T\mathbf{x}\mathbf{w}+\lambda\mathbf{w}^T\mathbf{w}\Big]\\ &= -\mathbf{y}^T\mathbf{x}-\mathbf{x}^T\mathbf{y}+2\mathbf{x}^T\mathbf{x}\mathbf{w}+2\lambda\mathbf{w}\\ &= 0\end{aligned}$$

$$w=\frac{1}{2}(\mathbf{x}^T\mathbf{x}+\lambda\mathbf{I})^{-1}(\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y})$$

iv. When $\lambda=0$, w = $\frac{1}{2}(\mathbf{x}^T\mathbf{x})^{-1}\Big[\mathbf{y}^T\mathbf{x}+\mathbf{x}^T\mathbf{y}\Big]\neq 0$.
Therefore, since $\lambda>0$, there doesn't exist such λ such that w = 0.

1_notebook

January 21, 2020

1 Problem 1

Use this notebook to write your code for problem 1. Some example code, and a plotting function for drawing decision boundaries, are given below.

[1]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
%matplotlib inline
```

1.0.1 Load the data:

[2]:
```
data = np.loadtxt('./data/problem1data1.txt')
X = data[:, :2]
y = data[:, 2]
```

1.0.2 The function make_plot below is a helper function for plotting decision boundaries; you should not need to change it.

[3]:
```
def make_plot(X, y, clf, title, filename):
    '''
    Plots the decision boundary of the classifier <clf> (assumed to have been␣
↪fitted
    to X via clf.fit()) against the matrix of examples X with corresponding␣
↪labels y.

    Uses <title> as the title of the plot, saving the plot to <filename>.

    Note that X is expected to be a 2D numpy array of shape (num_samples,␣
↪num_dims).
    '''
    # Create a mesh of points at which to evaluate our classifier
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    # binarize
    Z = np.where(Z > 0, np.ones(len(Z)), -1 * np.ones(len(Z)))

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8, vmin=-1, vmax=1)

    # Also plot the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(title)
    plt.savefig(filename)
    plt.show()
```

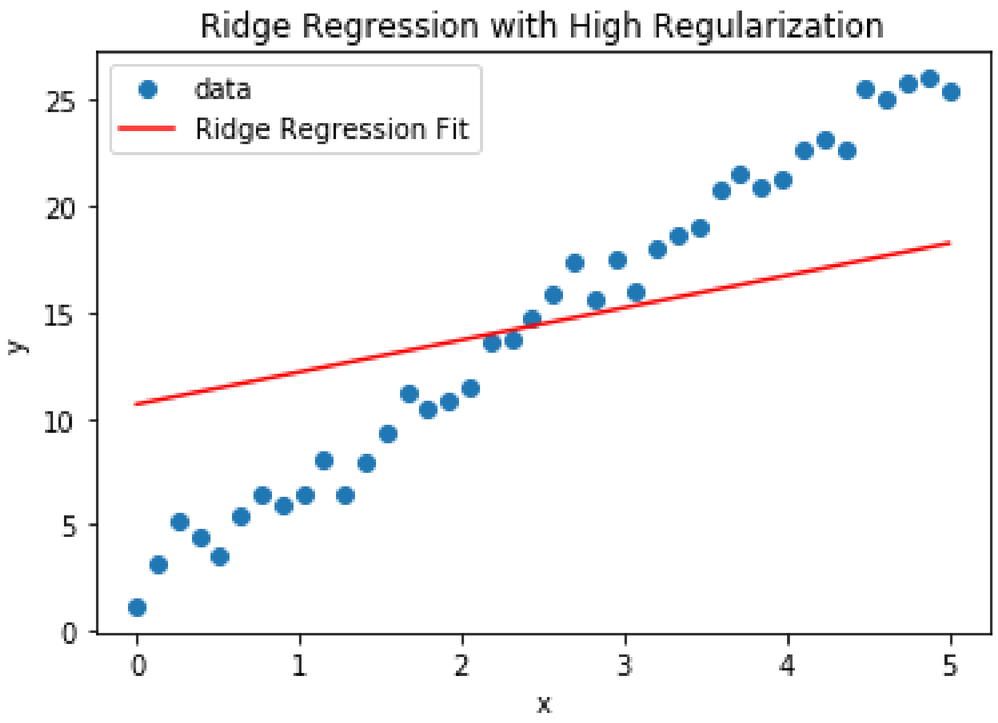
1.1 Here is some example code for performing regression with scikit-learn.

This section is not part of the problem! It demonstrates usage of the Ridge regression function, in particular illustrating what happens when the regularization strength is set to an overly-large number.

[6]:
```
# Instantiate a Ridge regression object:
ridge = Ridge(alpha = 200)

# Generate some fake data: y is linearly dependent on x, plus some noise.
n_pts = 40

x = np.linspace(0, 5, n_pts)
y = 5 * x + np.random.randn(n_pts) + 2

x = np.reshape(x, (-1, 1))   # Ridge regression function expects a 2D matrix

plt.figure()
plt.plot(x, y, marker = 'o', linewidth = 0)

ridge.fit(x, y)   # Fit the ridge regression model to the data
print('Ridge regression fit y = %fx + %f' % (ridge.coef_, ridge.intercept_))

# Add ridge regression line to the plot:
plt.plot(x, ridge.coef_ * x + ridge.intercept_, color = 'red')
plt.legend(['data', 'Ridge Regression Fit'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Ridge Regression with High Regularization')
```

```
Ridge regression fit y = 1.515800x + 10.675240
```

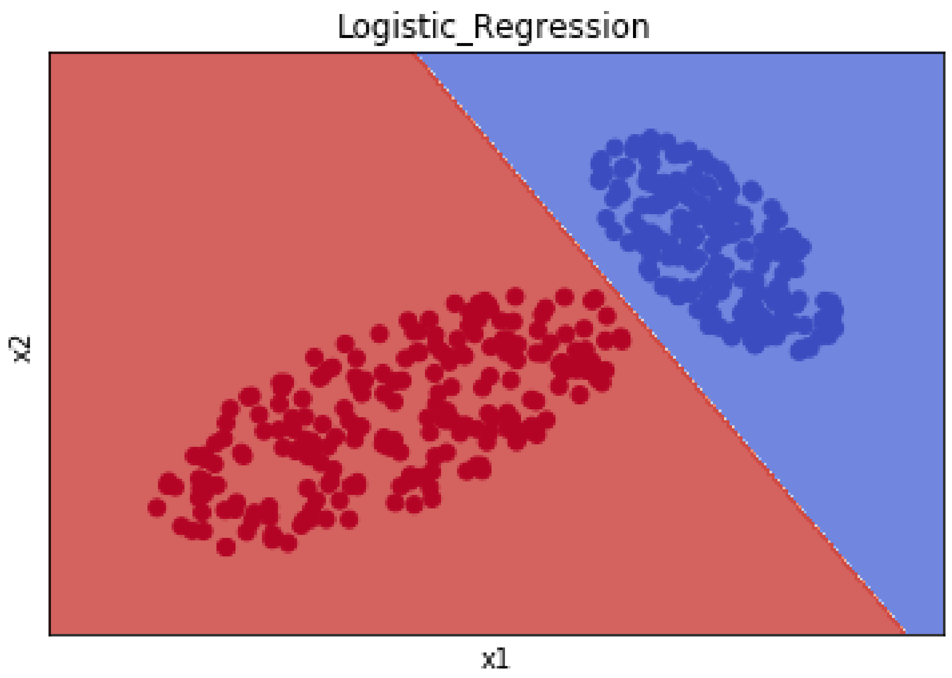
[6]:
```
Text(0.5, 1.0, 'Ridge Regression with High Regularization')
```



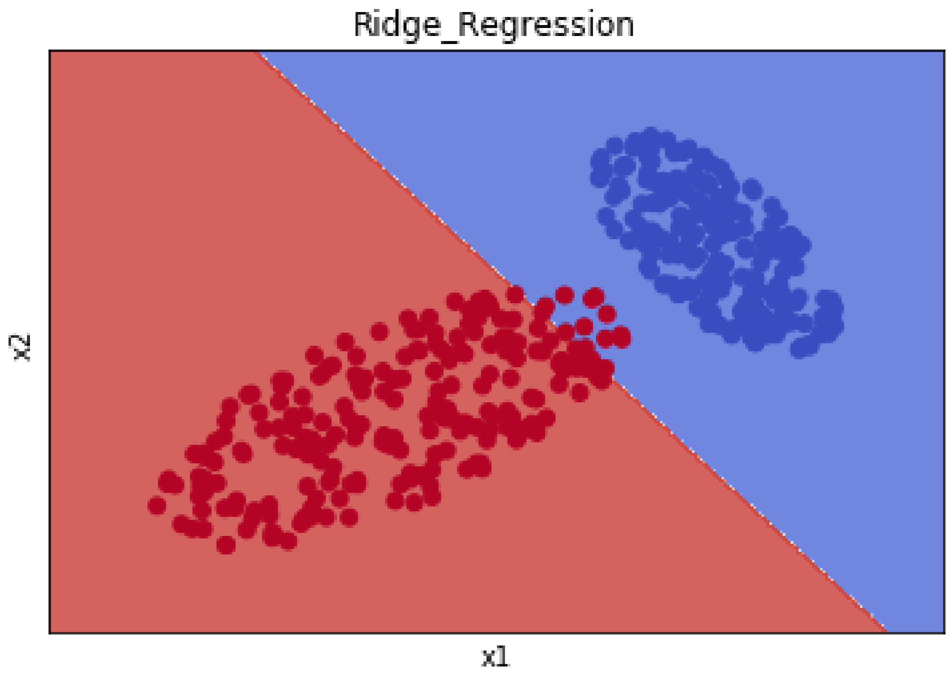
2 Your code for problem 1

[4]:
```
# C = Inverse of regularization strength; smaller values specify stronger␣
↪regularization.
clf = LogisticRegression(C = 1000)
clf.fit(X, y)
make_plot(X, y, clf, "Logistic_Regression", "Logistic_Regression")
```

```
C:\Users\Changhao\Anaconda3\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```



[5]:
```
# alpha = Regularization strength; larger values specify stronger␣
  ↪regularization.
# alpha corresponds to C^-1 in other linear models such as LogisticRegression
clf = Ridge(alpha = 0.001)
clf.fit(X, y)
make_plot(X, y, clf, "Ridge_Regression", "Ridge_Regression")
```


Ridge_Regression
x2
x1

2_notebook

January 21, 2020

1 Problem 2

Use this notebook to write your code for problem 2. You may reuse your SGD code from last week.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

The following function may be useful for loading the necessary data.

[2]:
```
def load_data(filename):
    '''
    Function loads data stored in the file filename and returns it as a numpy␣
 ↪ndarray.

    Inputs:
        filename: given as a string.

    Outputs:
        Data contained in the file, returned as a numpy ndarray
    '''
    return np.loadtxt(filename, skiprows=1, delimiter=',')
```

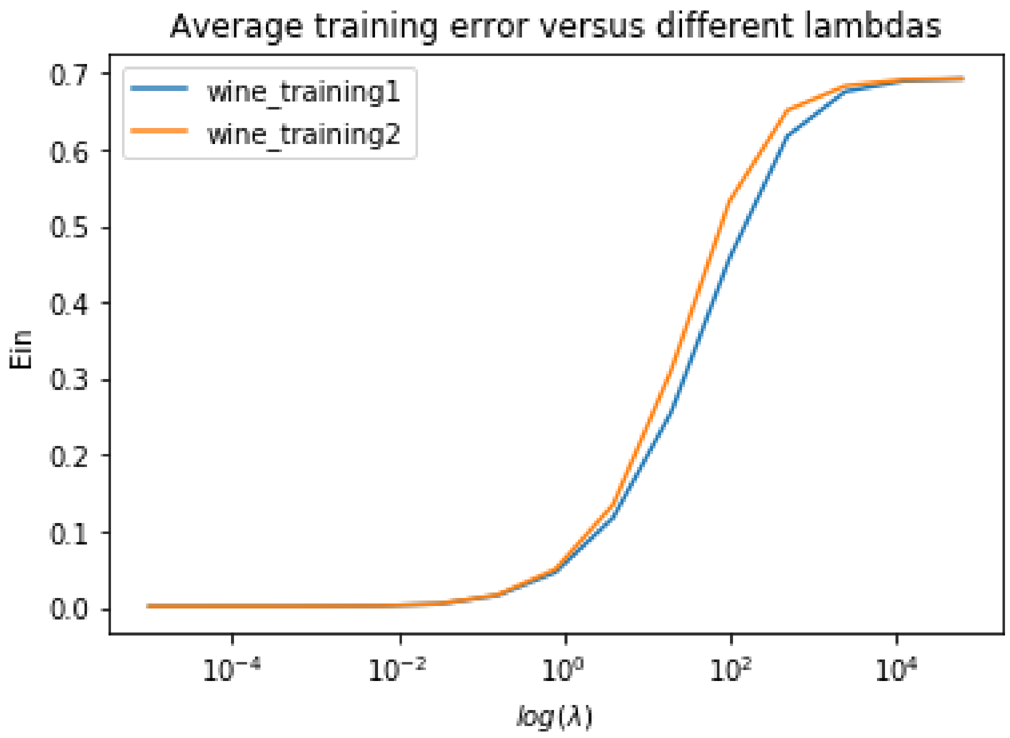
[3]:
```
def loss(X, Y, w):
    '''
    Calculate the log loss function.

    Inputs:
        X: A (N, D) shaped numpy array containing the data points.
        Y: A (N, ) shaped numpy array containing the (float) labels of the data␣
 ↪points.
        w: A (D, ) shaped numpy array containing the weight vector.

    Outputs:
        The loss evaluated with respect to X, Y, and w.
    '''
    log_loss = np.sum( [np.log(1 + np.exp( -Y[i] * np.dot(w, X[i]) ) ) for i in␣
 ↪range(X.shape[0])] )

    return log_loss

    pass

def gradient(x, y, w, lamda, N):
    '''
    Calculate the gradient of the loss function + regularization with respect␣
 ↪to
    a single point (x, y), and using weight vector w.

    Inputs:
        x: A (D, ) shaped numpy array containing a single data point.
        y: The float label for the data point.
        w: A (D, ) shaped numpy array containing the weight vector.
        lamda
        N

    Output:
        The gradient of the loss with respect to x, y, and w.
    '''

    log_gradient = (1 - 1 / (1 + (np.exp(-y * np.dot(w, x) ) ) ) ) * (-y) * x
    regularization = 2 * lamda * w / N
    total_gradient = log_gradient + regularization
    return total_gradient

    pass

def SGD(X, Y, w_start, eta, N_epochs, lamda):
    '''
    Perform SGD using dataset (X, Y), initial weight vector w_start,
    learning rate eta, and N_epochs epochs.

    Inputs:
        X: A (N, D) shaped numpy array containing the data points.
        Y: A (N, ) shaped numpy array containing the (float) labels of the data␣
 ↪points.
        w_start:  A (D, ) shaped numpy array containing the weight vector␣
 ↪initialization.
        eta: The step size.
        N_epochs: The number of epochs (iterations) to run SGD.

    Outputs:
        w: A (D, ) shaped array containing the final weight vector.
    '''
    w = w_start
    W = np.array([])

    for i in range(N_epochs):
        index = np.random.permutation(len(Y))
        W = np.append(W, w)
        for j in index:
            w = w - eta * gradient(X[j], Y[j], w, lamda, N = X.shape[0])

    return w

    pass
```

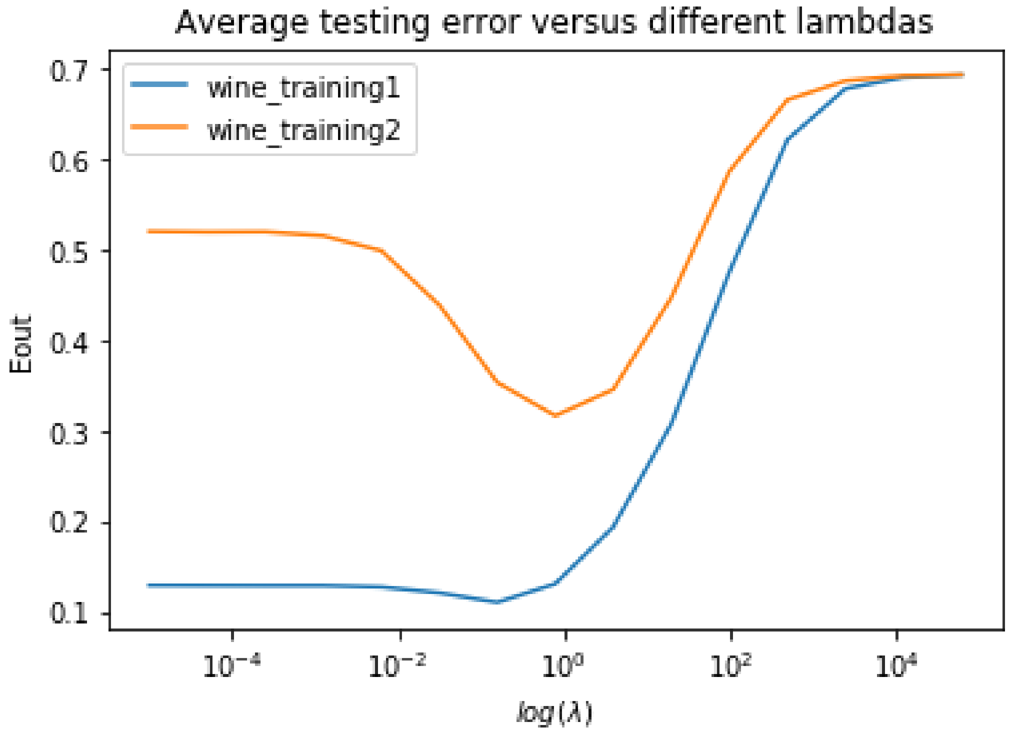
[4]:
```
def normalize(X): # normalize input data X to prevent numerical instability␣
↪issues (overflow or underflow)
    mean = np.array([])
    std = np.array([])

    for i in range(X.shape[1]):
        mean = np.append(mean, X[:, i].mean())
        std = np.append(std, np.std(X[:, i]))
        X[:, i] = (X[:, i] - mean[i]) / std[i]

    return X, mean, std
```

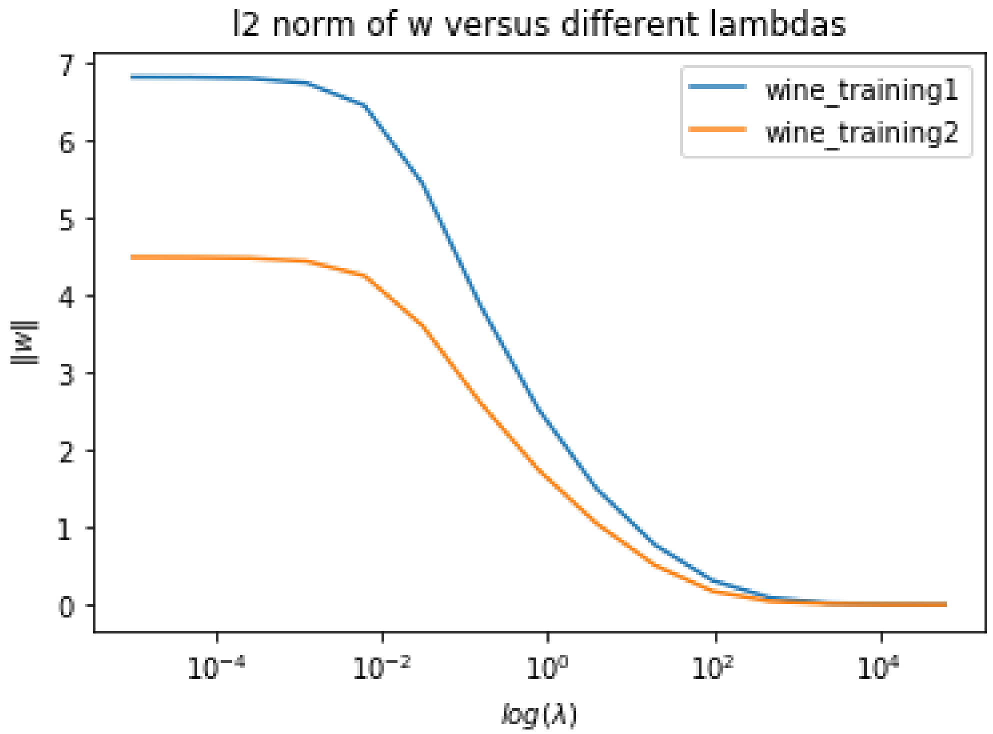
[5]:
```
def RegLogReg(X_train, y_train, X_test, y_test): # l2 regularized log␣
↪regression using SGD core
    N_epochs = 20000 # each epoch performs one SGD iteration for each point in␣
↪the training dataset
    eta = 5e-4
    lambdas = [0.00001]
    for i in range(14):
        lambdas.append(lambdas[i]*5)
    Ein = np.array([])
    Eout = np.array([])
    norm = np.array([])

    for lamda in lambdas:
        w_start = np.random.uniform(0, 0.01, 14) # initialize weights to small␣
↪random numbers
        W = SGD(X_train, y_train, w_start, eta, N_epochs, lamda)
        Ein = np.append(Ein, loss(X_train, y_train, W) / X_train.shape[0])
        Eout = np.append(Eout, loss(X_test, y_test, W) / X_test.shape[0])
        norm = np.append(norm, (np.linalg.norm(W)) )
    return lambdas, Ein, Eout, norm
```

[6]:
```
# load training data 1
training_data_1 = load_data('./data/wine_training1.txt')
X_train = training_data_1[:, 1:]
y_train_1 = training_data_1[:, :1]

X_train, mean1, std1 = normalize(X_train) # Normalize data before reshape␣
 ↪because bias term does not need to normalize
X_train_1 = np.c_[np.ones(X_train.shape[0]), X_train] # add one more column for␣
 ↪threshold

# load training data 2
training_data_2 = load_data('./data/wine_training2.txt')
X_train = training_data_2[:, 1:]
y_train_2 = training_data_2[:, :1]

X_train, mean2, std2 = normalize(X_train) # Normalize data before reshape␣
 ↪because bias term does not need to normalize
X_train_2 = np.c_[np.ones(X_train.shape[0]), X_train] # add one more column for␣
 ↪threshold

# load testing data normalized for training data 1
testing_data = load_data('./data/wine_testing.txt')
X_test = testing_data[:, 1:]
y_test_1 = testing_data[:, :1]

for i in range(X_test.shape[1]):
    X_test[:, i] = (X_test[:, i] - mean1[i]) / std1[i]
X_test_1 = np.c_[np.ones(X_test.shape[0]), X_test] # add one more column for␣
 ↪threshold

# load testing data normalized for training data 2
testing_data = load_data('./data/wine_testing.txt')
X_test = testing_data[:, 1:]
y_test_2 = testing_data[:, :1]

for i in range(X_test.shape[1]):
    X_test[:, i] = (X_test[:, i] - mean2[i]) / std2[i]
X_test_2 = np.c_[np.ones(X_test.shape[0]), X_test] # add one more column for␣
 ↪threshold
```

[7]:
```
lambdas_1, Ein1, Eout1, norm1 = RegLogReg(X_train_1, y_train_1, X_test_1,␣
 ↪y_test_1)
lambdas_2, Ein2, Eout2, norm2 = RegLogReg(X_train_2, y_train_2, X_test_2,␣
 ↪y_test_2)

plt.plot(lambdas_1, Ein1, label = 'wine_training1')
plt.plot(lambdas_2, Ein2, label = 'wine_training2')
plt.xscale('log')
plt.title('Average training error versus different lambdas')
plt.xlabel('$log(\lambda)$')
plt.ylabel('Ein')
plt.legend(loc = 'best')
plt.show()

plt.plot(lambdas_1, Eout1, label = 'wine_training1')
plt.plot(lambdas_2, Eout2, label = 'wine_training2')
plt.xscale('log')
plt.title('Average testing error versus different lambdas')
plt.xlabel('$log(\lambda)$')
plt.ylabel('Eout')
plt.legend(loc = 'best')
plt.show()

plt.plot(lambdas_1, norm1, label = 'wine_training1')
plt.plot(lambdas_2, norm2, label = 'wine_training2')
plt.xscale('log')
plt.title('l2 norm of w versus different lambdas')
plt.xlabel('$log(\lambda)$')
plt.ylabel('$\Vert w \Vert$')
plt.legend(loc = 'best')
plt.show()
```




Average testing error versus different lambdas
Eout
0.7
0.6
0.5
0.4
0.3
0.2
0.1
wine_training1
wine_training2
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
$log(\lambda)$


l2 norm of w versus different lambdas
$||w||$
7
6
5
4
3
2
1
0
wine_training1
wine_training2
10^{-4}
10^{-2}
10^{0}
10^{2}
10^{4}
$log(\lambda)$

3_notebook

January 21, 2020

1 Problem 3

Use this notebook to write your code for problem 3.

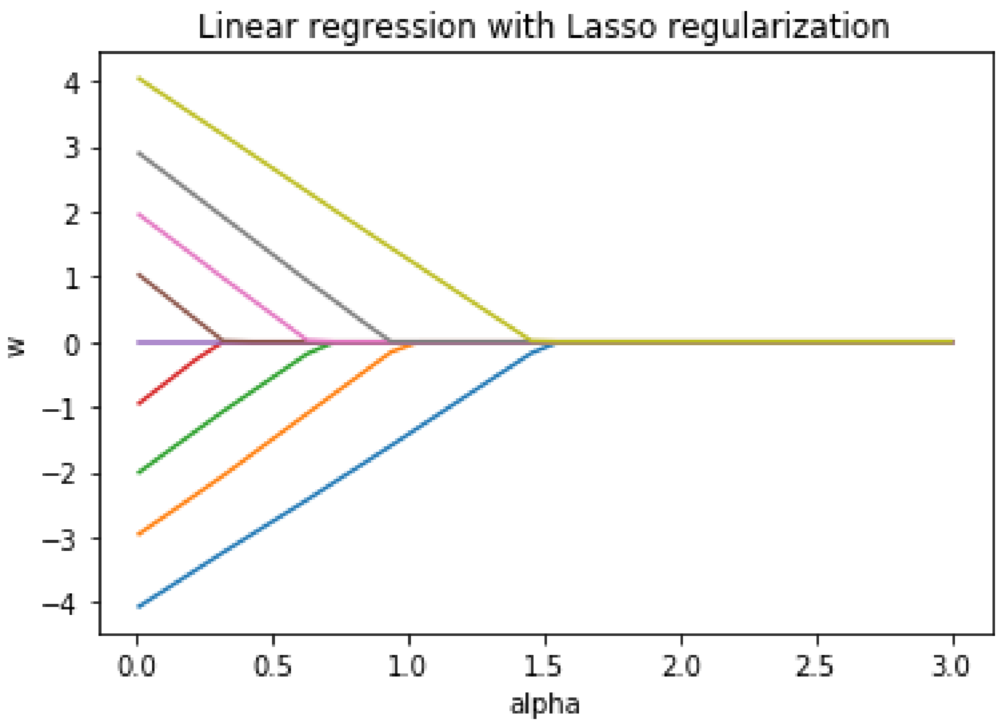
[1]:
```
import numpy as np
from matplotlib import pyplot as plt
from numpy import genfromtxt
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
%matplotlib inline
```

1.1 Load data

[2]:
```
train_file = './data/problem3data.txt'
train_data = genfromtxt(train_file, delimiter='\t')

y_train = train_data[:, 9]
X_train = train_data[:, :9]
```

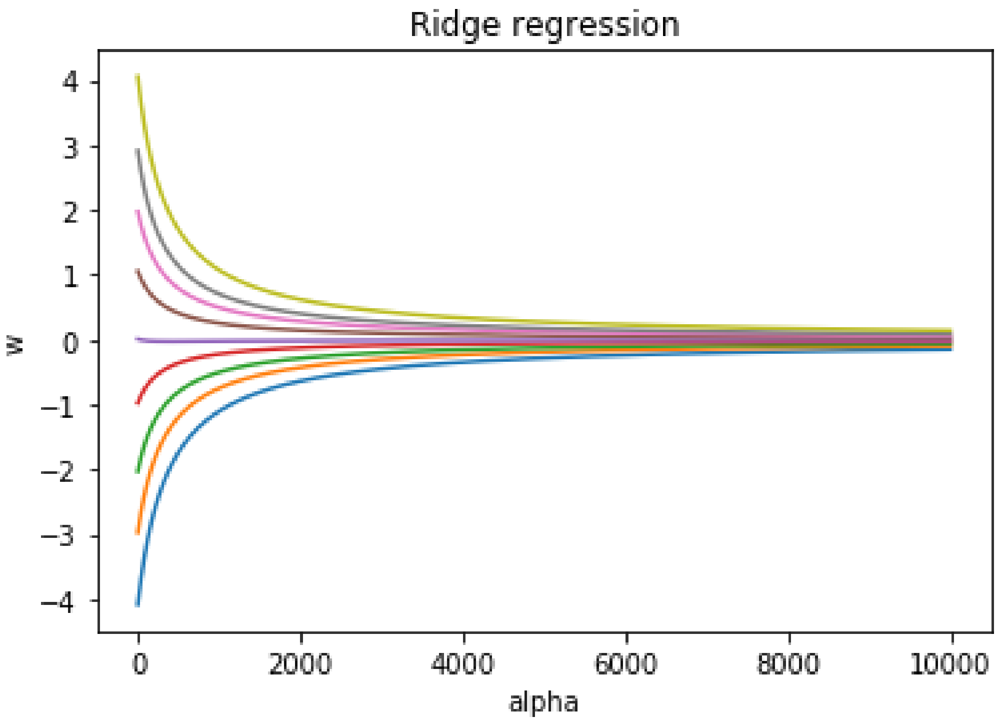
[3]:
```
# Linear regression with Lasso regularization
W_LRLasso = []
for i in np.linspace(0.01, 3, 30):
    clf = Lasso(alpha = i)
    clf.fit(X_train, y_train)
    w = clf.coef_
    W_LRLasso.append(w)

plt.plot(np.linspace(0.01, 3, 30), W_LRLasso)
plt.title("Linear regression with Lasso regularization")
plt.xlabel("alpha")
plt.ylabel("w")
```

[3]:
```
Text(0, 0.5, 'w')
```



```
[4]: # Ridge regression
     W_RRLasso = []
     for i in np.arange(1,10001):
         clf = Ridge(alpha = i)
         clf.fit(X_train, y_train)
         w = clf.coef_
         W_RRLasso.append(w)

     plt.plot(np.arange(1,10001), W_RRLasso)
     plt.title("Ridge regression")
     plt.xlabel("alpha")
     plt.ylabel("w")
```

```
[4]: Text(0, 0.5, 'w')
```


Ridge regression
w
alpha
4
3
2
1
0
−1
−2
−3
−4
0
2000
4000
6000
8000
10000