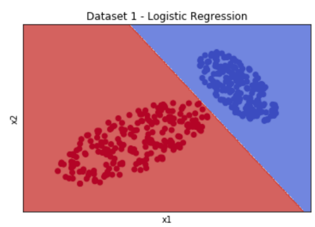
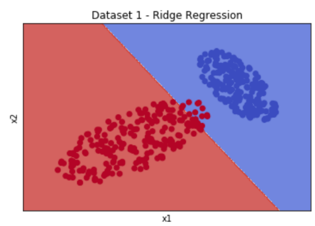
1 Comparing Different Loss Functions

Problem A:
Squared loss is often a terrible choice of loss function to train on for classification problems since it heavily penalizes outliers with high values even if the sign is accurate (the same).

Problem B:





Problem C:
Given the set of points and labels, we note that x_0 and y will be 1 and 1, respectively except for point S_3, which has -1 for the value of y. We calculate values for each point in $S = (.5, 3), (2, -2), (-3, 1)$ to get the following values:

$S_1 : (.5, 3)$

$$log = (-.39, -.19, -1.1)$$
$$hinge = (-1, -.5, -3)$$

$S_2 : (2, -2)$

$$log = (-.12, -.24, .24)$$
$$hinge = (0, 0, 0)$$

$S_3 : (-3, 1)$

$$log = (.05, -.14, .05)$$
$$hinge = (0, 0, 0)$$

Problem D:
As yw^tx progressively becomes bigger, the log loss gradient gets closer and closer to zero, as we can see given its loss function. The hinge loss gradient also converges to 0 if all points have been classified correctly since the hinge loss takes the max value between 0 and $1 - yw^Tx$. To order to reduce training error,

For a linearly separable dataset, is there any way to reduce or eliminate training error without changing decision boundary?

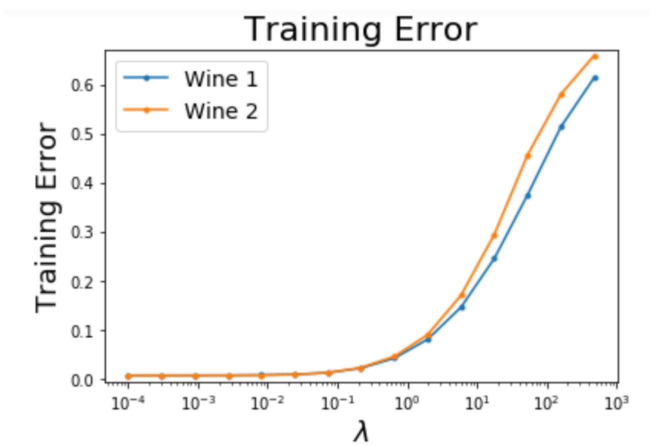
Problem E:
For an SVM to be a maximum margin classifier, its objective must not be to minimize just L_{hinge}, but to minimize $L_{hinge} + \lambda||w||^2$ for some λ greater than 0 because minimizing only L_{hinge} will only correctly classify all points without regard to margin.

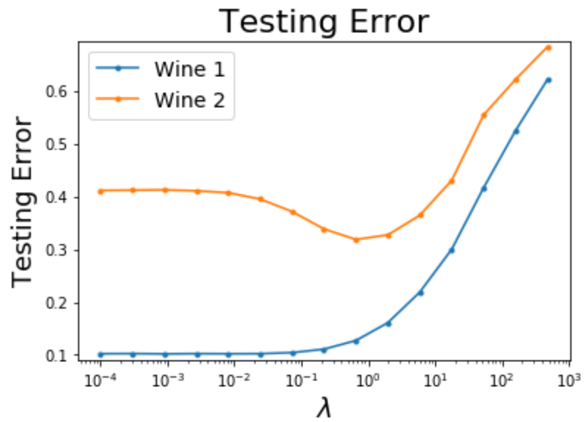
2 Effects of Regularization

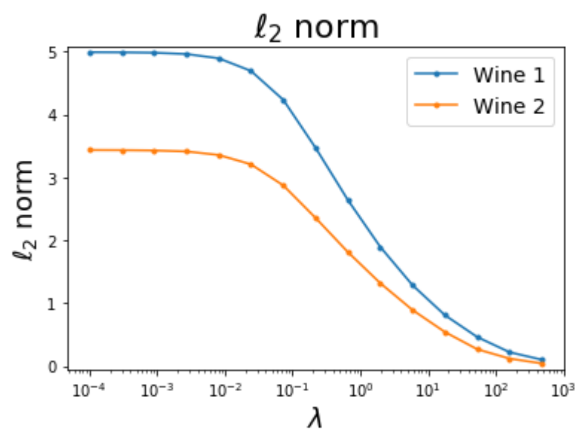
Problem A:
In order to prevent overfitting in the least-squares regression problem, we add a regularization penalty term. Adding this penalty term cannot decrease the training (in-sample) error because the optimal way to minimize this in-sample error is through using the closed-form solution. Adding this penalty term cannot guarantee the decrease of the testing (out-sample) error either because we can both underfit and overfit—overfitting would reduce variance and improve generalization and conversely underfitting could increase bias with the reduction of variance.

Problem B:
l_0 regularization is rarely used because it isn't continuous and it isn't convex, which can't be used alongside gradients or derivatives to minimize error.

Problem C:







Problem D:
Given that the data in `wine−training2.txt` is a subset of the data in `wine−training1.txt`, if we compare results from the `training1` (100 pts) vs `training2` (40 pts) in training error, we see that `training1` has less error than `training2` as λ increases. This makes sense because `training2` has smaller data and at large λ models, that is more detrimental to the training error of smaller sets.

In our second graph, we can see that `training2` has been overfitted—testing error increases much faster with increasing λ in comparison to `training1`. This makes sense since we know that testing error should be lower in `training1` since we have more training data to decrease variance.

Problem E:
Examining the qualitative behavior of the training and test errors with different λs while training with data in `training1` reveals to us that after a certain threshold ($\lambda > 1$), there is an exponential growth in both training and testing error. This indicates that at large λ, the

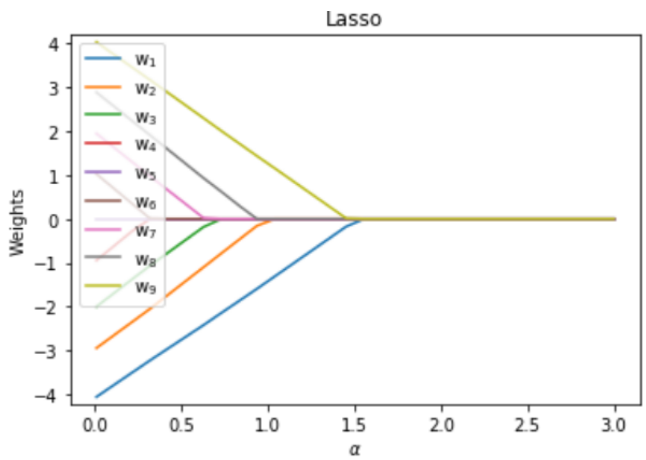
model fails to accurately represent our given data.

Problem F:
Examining the qualitative behavior of the l_2 norm with different λs in `training1` shows us that as λ increases, the norm decreases (also the case for `training2`). This matches our understanding that λ increasing places more weight onto the regularization term and thus decreases the l_2 norm.

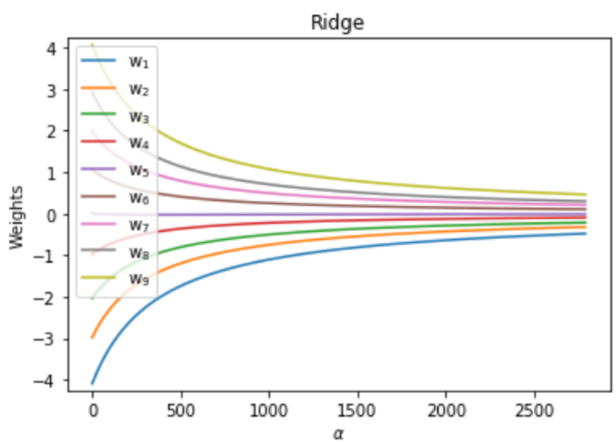
Problem G:
If the model were trained with `training2`, I would choose a λ that has a value of about 1 (of course, we would calculate the minimum of the curve in Graph 2) because that seems to be where testing error is minimized, as we see in the second graph.

3 Lasso vs Ridge Regularization

Problem A:
i.



ii.



iii. For lasso regression, as the regularization parameter increases, the number of 0 model weights also increases. For ridge regression, these weights approach 0 but do not become 0.

Problem B:
i. We take the subgradient to get the following:

$$= -2X^T(y - X^T w) + \lambda = 0$$
$$w = -(\lambda - 2X^T y)(2X^T X)^{-1}$$

ii. w_1 approaches 0 when $\lambda = ||2X^T y||$

iii.

$$
\begin{aligned}
&= -2X^T(y - X^T w) + 2\lambda w = 0 \\
w &= X^T y(\lambda I + X^T X)^-1
\end{aligned}
$$

iv. There can't exist a $\lambda > 0$ s.t. $w_i = 0$ because if there were, then manipulating our equation by multiplying $\lambda I + X^T X$ on both sides would give us $0 = X^T y$. Now this shows independence from λ, relating back to how Ridge weights only approach 0.

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.

In [3]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
%matplotlib inline
```

Load the data:

In [4]:
```
data = np.loadtxt('data/problem1data1.txt')
X = data[:, :2]
Y = data[:, 2]
```

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

In [5]:
```
def make_plot(X, y, clf, title, filename):
    '''
    Plots the decision boundary of the classifier <clf> (assumed to have been fitt
ed
    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_dim
s).
    '''
    # 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()
```

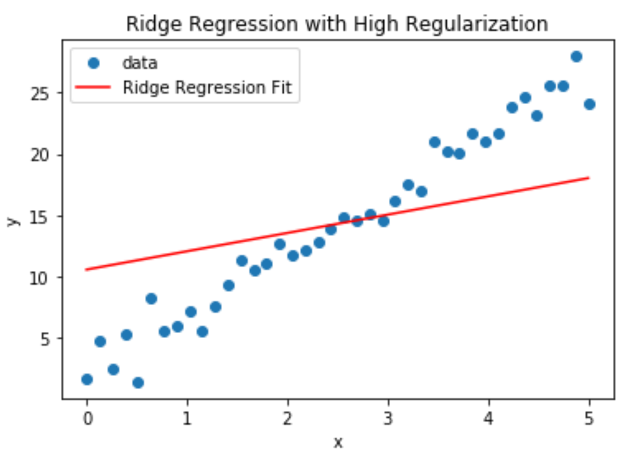
Here is some example code for performing regression with scikit-learn.

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.

In [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')
```

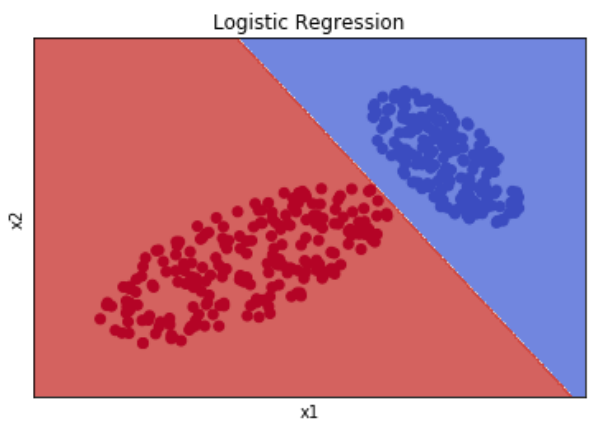
```
Ridge regression fit y = 1.487240x + 10.580993
```

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



Your code for problem 1

In [8]:
```
#=============================================
# TODO: Implement your code for Problem 1 here.
# Use as many cells as you need.
#=============================================

def logistic(x, y):
    # return decision boundary of classifier & predicted values
    clf = LogisticRegression()
    clf = clf.fit(x, y)
    predicted = clf.predict(x)
    return clf, predicted

def ridge(x, y):
    # run ridge and return weights
    clf = Ridge(alpha = 200)
    clf.fit(x, y)
    predicted = clf.predict(x)
    return clf, predicted

clf_log, p_log = logistic(X, Y)
make_plot(X, Y, clf_log, "Logistic Regression", "p_log")

clf_ridge, p_ridge = ridge(X, Y)
make_plot(X, Y, clf_ridge, "Ridge Regression", "p_ridge")
```



In [ ]:

Problem 2

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

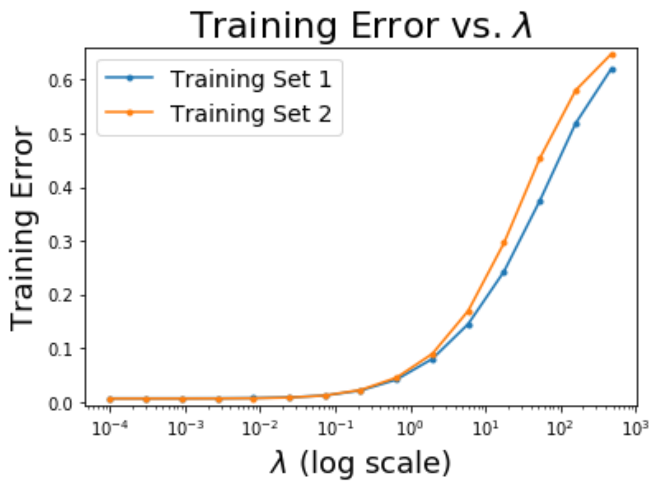
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import random
%matplotlib inline
```

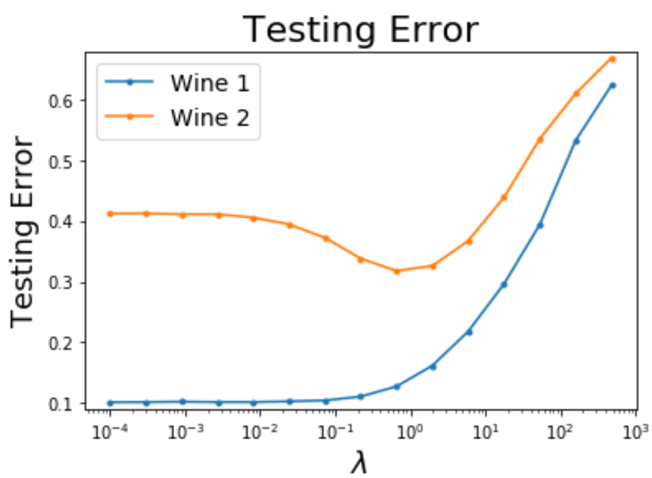
The following function may be useful for loading the necessary data.

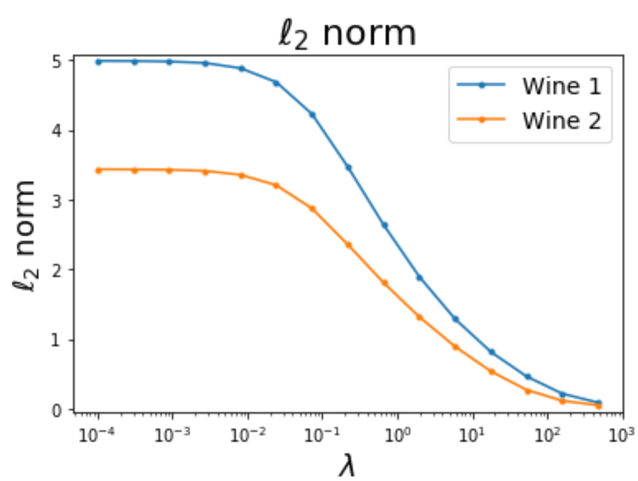
In [ ]:
```
def load_data(filename):
    return np.loadtxt(filename, skiprows=1, delimiter=',')

def normalX(x):
    xN = x
    means = []
    stds = []
    for i in range(1, len(xN[0])):
        means.append(np.mean(xN[:,[i]]))
        stds.append(np.std(xN[:,[i]]))
    for i in inputsNormalized:
        for j in range(1, len(i)):
            i[j] = (i[j] - means[j-1]) / stds[j-1]
    return xN

def normalY(x, y):
    xN = x
    yN = y
    means = []
    stds = []
    for i in range(1, len(xN[0])):
        means.append(np.mean(xN[:,[i]]))
        stds.append(np.std(xN[:,[i]]))
    for i in yN:
        for j in range(1, len(i)):
            i[j] = (i[j] - means[j-1]) / stds[j-1]
    return yN

def getXYnorm(data):
    x = []
    y = []
    arr = np.asarray([1.0])
    for i in data:
        x.append(np.concatenate((arr, i[1:]), axis = 0))
        y.append(i[0])
    return normalX(np.asarray(x)), np.asarray(y)

def getXY(data):
    x = []
    y = []
    arr = np.asarray([1.0])
    for i in data:
        x.append(np.concatenate((arr, i[1:]), axis = 0))
        y.append(i[0])
    return np.asarray(x), np.asarray(y)

def loss(weights, y, x):
    totalLoss = 0
    for i in range(len(x)):
        if (y[i] == -1):
            add = np.log(1 / (1 + math.exp(np.inner(weights, x[i]))))
        else:
            add = np.log(1 / (1 + math.exp(-np.inner(weights, x[i]))))
        totalLoss += add
    return totalLoss / len(x) * -1
```

In [ ]:
```
def calcGrad(x, y, w, l, size):
    ret =  2* l * w / size - x * y / (math.exp(np.inner(w, x) * y) + 1)
    return ret

def L2Norm(w):
    return math.sqrt((np.inner(w, w)))

def runSGD(data, iW, stepSize, l):
    numEpochs = 20000
    totalLoss = []
    w = iW
    x, y = getXYnorm(data)
    currLoss = loss(w, y, x)
    totalLoss.append(currLoss)

    for _ in range(numEpochs):
        np.random.shuffle(data)
        x, y = getXYnorm(data)

        for i in range(len(x)):
            grad = calcGrad(x[i], y[i], w, l, len(x))
            w -= stepSize * grad

        currLoss = loss(w, y, x)
        totalLoss.append(currLoss)

    return w, totalLoss

# wine 1 -------------
data1 = load_data("data/wine_training1.txt")
lambda0 = 0.00001
lambdas = []
w = []
loss = []
step = math.exp(-4)

for i in range(15):
    start = [0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
0.001, 0.001, 0.001, 0.001]
    finWeights, totalLoss = runSGD(data1, start, step, lambda0)
    w.append(finWeights)
    loss.append(totalLoss[-1])
    lambdas.append(lambda0)
    lambda0 *= 3

# wine 2 -------------
lambdas2 = []
w2 = []
loss2 = []
lambda0 = 0.00001
step = math.exp(-4)
data2 = load_data("data/wine_training2.txt")
```

In [29]:
```
for i in range(15):
    start = [0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001,
0.001, 0.001, 0.001, 0.001]
    finWeights2, totalLoss2 = runSGD(data2, start, step, lambda0)
    w2.append(finWeights2)
    loss2.append(totalLoss2[-1])
    lambdas2.append(lambda0)
    lambda0 *= 3

fig = plt.figure()
plt.title(r'Training Error vs. $\lambda$', fontsize = 22)
plt.plot(lambdas, loss, lambdas1, loss1, marker = '.')
plt.legend(('Training Set 1', 'Training Set 2'), loc = 'best', fontsize = 14)
plt.xscale('log')
plt.xlabel('$\lambda$ (log scale)', fontsize = 18)
plt.ylabel('Training Error', fontsize = 18)
plt.margins(y=0.02)

# test -------------
trainx1, trainy1 = getXY(allData)
trainx2, trainy2 = getXY(allData1)
testData = load_data("data/wine_testing.txt")
testx1, testy1 = getXY(testData)
testx2, testy2 = getXY(testData)

testerr1 = []
testerr2 = []

testxnorm1 = normalY(trainx1, testx1)
testxnorm2 = normalY(trainx2, testx2)

for i in w:
    testerr1.append(loss(i, trainy1, testxnorm1))
for j in w2:
    testerr2.append(loss(j, trainy1, testxnorm2))

fig = plt.figure()
plt.title(r'Testing Error')
plt.plot(lambdas, testerr1, lambdas1, testerr2, marker = '.')
plt.legend(('Wine 1', 'Wine 2'))
plt.xscale('log')
plt.xlabel('$\lambda$')
plt.ylabel('Testing Error')
plt.margins(y=0.02)

# lambda -------------
norm1 = []
norm2 = []

for i in w:
    norm1.append(L2Norm(i))
for j in w2:
    norm2.append(L2Norm(j))

fig = plt.figure()
plt.title(r'$\ell_2$ norm')
plt.plot(lambdas, norm1, lambdas1, norm2, marker = '.')
plt.legend(('Wine 1', 'Wine 2'))
plt.xscale('log')
plt.xlabel('$\lambda$')
plt.ylabel('$\ell_2$ norm')
plt.margins(y=0.02)
```


Training Error vs. λ
Training Set 1
Training Set 2
Training Error
λ (log scale)


Testing Error
Wine 1
Wine 2
Testing Error
λ


ℓ_2 norm
Wine 1
Wine 2
ℓ_2 norm
λ

In [ ]:

In [ ]:

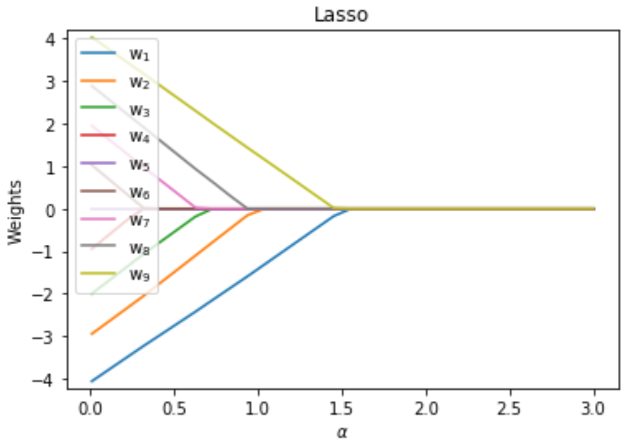
Problem 3

Use this notebook to write your code for problem 3.

In [26]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
%matplotlib inline
```

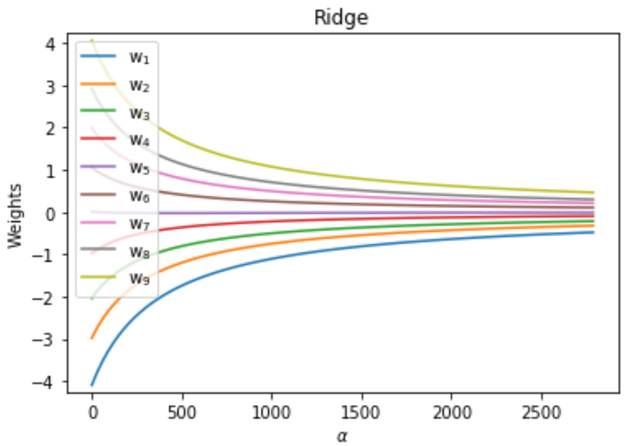
Load data

In [43]:
```
train_file = 'data/problem3data.txt'
train_data = genfromtxt(train_file, delimiter='\t')

y_train = train_data[:, 9]
x_train = train_data[:, :9]

def lasso(a, x, y):
    # run lasso and return weights
    clf = Lasso(alpha=a)
    clf.fit(x, y)
    return clf.coef_

def ridge(a, x, y):
    # run ridge and return weights
    clf = linear_model.Ridge(alpha=a)
    clf.fit(x, y)
    return clf.coef_

c = []
c1 = []
c2 = []
c3 = []
c4 = []
c5 = []
c6 = []
c7 = []
c8 = []
c9 = []

alphas = np.linspace(.01, 3, 30)

for alpha in alphas:
    c.append(lasso(alpha, x_train, y_train))

for i in c:
    c1.append(i[0])
    c2.append(i[1])
    c3.append(i[2])
    c4.append(i[3])
    c5.append(i[4])
    c6.append(i[5])
    c7.append(i[6])
    c8.append(i[7])
    c9.append(i[8])

x = alphas
fig = plt.figure()
plt.title('Lasso')
plt.plot(x, c1, x, c2, x, c3, x , c4, x, c5, x, c6, x, c7, x, c8, x, c9)

plt.legend(('w$_1$','w$_2$', 'w$_3$', 'w$_4$', 'w$_5$', 'w$_6$', 'w$_7$', 'w$_8$',
'w$_9$'))
plt.xlabel(r'$\alpha$')
plt.ylabel('Weights')
plt.margins(y=0.02)
```


Lasso
Weights
α
w1
w2
w3
w4
w5
w6
w7
w8
w9
4
3
2
1
0
−1
−2
−3
−4
0.0
0.5
1.0
1.5
2.0
2.5
3.0

In [63]:

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

y_train = train_data[:, 9]
x_train = train_data[:, :9]

c = []
c1 = []
c2 = []
c3 = []
c4 = []
c5 = []
c6 = []
c7 = []
c8 = []
c9 = []

alphas = []
alpha = 0.0
n = 0

while n < 6:
    c.append(ridge(alpha, x_train, y_train))
    alphas.append(alpha)
    alpha += 30

    for i in c[-1]:
        if math.fabs(i) < 0.3:
            n += 1

for i in c:
    c1.append(i[0])
    c2.append(i[1])
    c3.append(i[2])
    c4.append(i[3])
    c5.append(i[4])
    c6.append(i[5])
    c7.append(i[6])
    c8.append(i[7])
    c9.append(i[8])

x = alphas
fig = plt.figure()
plt.title('Ridge')
plt.plot(x, c1, x, c2, x, c3, x , c4, x, c5, x, c6, x, c7, x, c8, x,c9)
plt.legend(('w$_1$','w$_2$', 'w$_3$', 'w$_4$', 'w$_5$', 'w$_6$', 'w$_7$', 'w$_8$',
'w$_9$'))
plt.xlabel(r'$\alpha$')
plt.ylabel('Weights')
plt.margins(y=0.02)
```


Ridge
Weights
α
w_1
w_2
w_3
w_4
w_5
w_6
w_7
w_8
w_9
4
3
2
1
0
−1
−2
−3
−4
0
500
1000
1500
2000
2500

In [ ]: