HW3_problem4

April 29, 2015

```
In [1]: %matplotlib inline
        import scipy
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: data_link = "http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/prostate.data"
        import requests
        from cStringIO import StringIO
        data = StringIO(requests.get(data_link).text)
        df = pd.read_csv(data, sep="\t", index_col=0)
```

```
In [3]: df.head()
```

Out[3]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa	train
1	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	-0.430783	T
2	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	-0.162519	T
3	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	-0.162519	T
4	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	-0.162519	T
5	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	0.371564	T

```
In [10]: x = df[["lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason", "pgg45"]]
         y = df["lpsa"]
         mask=(df.train=="T").to_dense()
```

```
In [11]: from scipy.optimize import linprog
```

Generic Linear regression is given by

$$XA + b = Y$$

Where X is the feature matrix of size (m x n) and Y is the value matrix (m x 1) to be predicted.

The minimization equation is

$$min(|Y - XA - b|)$$

this in LP can be written as

$$\text{Assuming,} e_i \text{ denotes the error for each element, then} \tag{1}$$
$$e_i = y_i - x_i A - b \tag{2}$$
$$\text{LP Equation is} \tag{3}$$
$$min(\sum_i e_i) \tag{4}$$
$$\text{Such That } (XA + b - Y) \leq e \rightarrow XA + b - e \leq Y \tag{5}$$
$$-(XA + b - Y) \leq e \rightarrow -(XA + b + e) \leq -Y \tag{6}$$

In order to use a linear program solver, we convert the above equation and constraints to canonical form. The canonical form is as follows

$$min(C^T * e')\text{such that} \tag{7}$$
$$AX + b * 1_{\text{mx1}} - eI < Y \tag{8}$$
$$-AX - b * 1_{\text{mx1}} - eI < -Y \tag{9}$$
$$\tag{10}$$

Here e' is a matrix of dimensions (n+m+1) x (1)

thus the constraints can be re-written in terms of e' by stacking matrices A,I, $1_{\text{m x n}}$. For example, the first constraint can be written as $X'e' < Y$, where X' is a matrix of (m, m+n+1) dimensions. The mxn elements is the X matrix, the elements from columns (m+1..n) is an Identity matrix and the last column (m+n+1) is full of ones.

C is a vector of (m +n +1) dimensions where the first all elements except (m+1..n) are zero

```
In [138]: from scipy.optimize import linprog
          from sklearn.base import RegressorMixin
          class linearReg(RegressorMixin):
              def __init__(self):
                  self.b = None
                  self.A = None
                  self.lp_result = None

              def fit(self, X, y):
                  m, n = X.shape
                  C = np.vstack((np.zeros((n,1)), np.ones((m,1)), 0)) # Create C by stacking the zeros
                  X_dash = np.hstack((X, -1*np.identity(m), np.ones((m,1))))
                  X_dash2 = np.hstack((-1*X, -1*np.identity(m), -1*np.ones((m,1))))
                  #stack both these matrixes vertically so that both constraints can be checked in one
                  lhs = np.vstack((X_dash, X_dash2))
                  #similarly combine both RHS of the constraints
                  rhs = np.vstack((y, -y))
                  self.lp_result = linprog(C.flatten(), lhs, rhs)
                  res = self.lp_result.x
                  self.A = res[0:n]
                  self.b = res[-1]
                  return self

              def predict(self, X):
                  xd = np.hstack((X, np.ones((X.shape[0],1))))
                  w = np.vstack((self.A[:, np.newaxis], self.b))
                  return np.dot(xd, w).flatten()
```

In [139]:
```
lpfit =linearReg().fit(x,y)
print x.shape
```

```
(97, 8)
```

1 comparing the performance of our regression with l2 optimized linear regression from sklearn

In [143]:
```
from sklearn.linear_model import LinearRegression

def l2_error(model, X, y):
    ypred = model.predict(X)
    return np.mean((y-ypred)**2)

lp_lr = linearReg().fit(x[mask], y[mask])
l2_lr = LinearRegression().fit(x[mask], y[mask])

print "l2 optimized linear regression, Train Error = ", l2_error(l2_lr, x[mask], y[mask])
print "linear programming Train Error", l2_error(lp_lr, x[mask], y[mask])

print "l2 optimized linear regression, Train Error = ", l2_error(l2_lr, x[~mask], y[~mask])
print "linear programming Train Error", l2_error(lp_lr, x[~mask], y[~mask])
```

```
l2 optimized linear regression, Train Error =  0.439199768058
linear programming Train Error 0.49702668751
l2 optimized linear regression, Train Error =  0.521274005508
linear programming Train Error 0.558719378544
```

In [ ]: