Logistic Regression

October 12, 2015

In [1]:
```
import numpy
import urllib
import scipy.optimize
import random
from math import exp
from math import log

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

print "Reading data..."
data = list(parseData("file:///users/YW/Dropbox/UCSD/FA2015/CSE255/HW/hw1/book_descriptions_500
print "done"

def inner(x,y):
  return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
  return 1.0 / (1 + exp(-x))

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
  loglikelihood = 0
  for i in range(len(X)):
    logit = inner(X[i], theta)
    loglikelihood -= log(1 + exp(-logit))
    if not y[i]:
      loglikelihood -= logit
  for k in range(len(theta)):
    loglikelihood -= lam * theta[k]*theta[k]
  print "ll =", loglikelihood
  return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
  dl = [0.0]*len(theta)
  for i in range(len(X)):
    # Fill in code for the derivative
    logit = inner(X[i], theta)
    for k in range(len(dl)):
        dl[k] += (1-sigmoid(logit))*X[i][k]
        if not y[i]:
            dl[k] -= X[i][k]
  for k in range(len(dl)):
    dl[k] -= 2*lam*theta[k]
  # Negate the return value since we're doing gradient *ascent*
  return numpy.array([-x for x in dl])

D_child = [d for d in data if "Children's Books" in d['categories']]
D_notchild =\
  [d for d in data if not("Children's Books" in d['categories'])][:len(D_child)]

data = D_child + D_notchild
random.seed(0)
random.shuffle(data)

X = [[1, "child" in d['description'], "magic" in d['description'], "funny" in d['description']]
y = ["Children's Books" in d['categories'] for d in data]

X_train = X[:len(X)/2]
X_valid = X[len(X)/2:3*len(X)/4]
X_test = X[3*len(X)/4:]

y_train = y[:len(y)/2]
y_valid = y[len(y)/2:3*len(y)/4]
y_test = y[3*len(y)/4:]

theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]), fprime, args = (X_train, y_train,
print "Final log likelihood =", -l
```

```
Reading data...
done
ll = -2872.40191624
ll = -2536.91244544
ll = -2307.88483356
ll = -2302.59095342
ll = -2301.68433071
ll = -2300.83583575
ll = -2298.83652773
ll = -2297.26637853
ll = -2297.0754507
ll = -2297.07034149
ll = -2297.07010839
ll = -2297.0700787
ll = -2297.06998251
ll = -2297.07002084
ll = -2297.06998075
Final log likelihood = -2297.06998075
```

```
In [2]: from sklearn import linear_model
        logreg = linear_model.LogisticRegression(fit_intercept=False)
```

```
In [3]: logreg.fit(X_train, y_train)
```

```
Out[3]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=False,
                  intercept_scaling=1, max_iter=100, multi_class='ovr',
                  penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                  verbose=0)
```

```
In [4]: print logreg.coef_
        print theta

[[-0.89569608  2.34170254  0.94704375  0.21062554]]
[-0.88852315  2.3260095   0.91260868  0.20626077]

In [5]: my_train_predictions = numpy.array([(True if inner(d, theta)>=0 else False) for d in X_train])

In [6]: my_test_predictions = numpy.array([(True if inner(d, theta)>=0 else False) for d in X_test])

In [7]: my_valid_predictions = numpy.array([(True if inner(d, theta)>=0 else False) for d in X_valid])

In [8]: train_predictions = logreg.predict(X_train)
        test_predictions = logreg.predict(X_test)
        valid_predictions = logreg.predict(X_valid)

In [9]: print 'My Train Accuracy:%.3f, My Test Accuracy:%.3f, My Valid Accuracy:%.3f'%\
        (1-numpy.mean(my_train_predictions!=y_train),
        1-numpy.mean(my_test_predictions!=y_test),
        1-numpy.mean(my_valid_predictions!=y_valid))
        print 'Train Accuracy:%.3f, Test Accuracy:%.3f, Valid Accuracy:%.3f'%\
        (1-numpy.mean(train_predictions!=y_train),
        1-numpy.mean(test_predictions!=y_test),
        1-numpy.mean(valid_predictions!=y_valid))

My Train Accuracy:0.749, My Test Accuracy:0.729, My Valid Accuracy:0.749
Train Accuracy:0.749, Test Accuracy:0.729, Valid Accuracy:0.749

In [10]: print 1-logreg.score(X_train, y_train)
         print 1-logreg.score(X_test, y_test)
         print 1-logreg.score(X_valid, y_valid)

0.250965250965
0.271235521236
0.250965250965

In [ ]:
```