CSE255 Homework1 Answer

Yue Wang, A53102167

October 12, 2015

Regression

1

The fitted values are $\theta_0 = 3.11521115$, $\theta_1 = 0.10905507$

Code Snippets:

```
def feature(datum):
  feat = [1]
  feat.append(datum['beer/ABV'])
  return feat
X = [feature(d) for d in data]
y = [d['review/taste'] for d in data]
theta, residuals, rank, s = np.linalg.lstsq(X, y)
print 'Theta0:%s, Theta1:%s'%(theta[0], theta[1])
```

2

degree:0

Train MSE:0.5135699375

coefficients:[3.92225]

degree:1

Train MSE:0.449696640735

coefficients:[3.11521115 0.10905507]

degree:2

Train MSE:0.443806278311

coefficients:[2.80999028 0.18846696 -0.00469694]

degree:3

Train MSE:0.438136515763

coefficients:[2.04075456e+00 4.23029268e-01 -2.35342452e-02 3.10155744e-04]

degree:4

Train MSE:0.43645058724

coefficients:[1.56837518e+00 5.96499090e-01 -4.34607088e-02 1.12759878e-03 -9.42560348e-06]

degree:5

Train MSE:0.436425509067

coefficients:[1.40998006e+00 6.71979699e-01 -5.57829661e-02 1.95198865e-03 -3.03848616e-05 1.73376523e-07]

```
def features(datum, degree):
    feat = [1]
    for i in xrange(1, degree+1):
        feat.append(datum['beer/ABV']**i)
    return feat
deg = 0
while True:
    regr = linear_model.LinearRegression(fit_intercept=False)
    X = [features(d, deg) for d in data]
    y = [d['review/taste'] for d in data]
    regr.fit(X, y)
    thisTrainError = np.mean((regr.predict(X)-y) ** 2)
    print "degree:%s\\\\"%(len(X[0])-1)
    print "Train MSE:%s\\\\"% thisTrainError
    print "coefficients:%s\\\\"% regr.coef_
    deg = deg + 1
    if deg > 5:
        break
```

3

Degree:0

Train:0.55835536

Test:0.46912512

Coef:[3.9092]

Degree:1

Train:0.483983105115

Test:0.423776528023

Coef:[2.99503282 0.11690802]

Degree:2

Train:0.471743067557

Test:0.427256126

Coef:[2.62007309 0.20716481 -0.00496806]

Degree:3

Train:0.457832195847

Test:0.432820707084

Coef:[1.57847740e+00 5.19869113e-01 -2.97470415e-02 3.97626061e-04]

Degree:4

Train:0.451641221709

Test:0.438301732573

Coef:[7.17629022e-01 8.26164532e-01 -6.32189636e-02 1.67076516e-03 -1.40075070e-05]

Degree:5

Train:0.451335575196

Test:0.439820774965

Coef:[1.16389773e+00 6.12905604e-01 -2.85943943e-02 -6.28209940e-04 4.41325901e-05 -4.79191916e-07]

Degree:6

Train:0.450769302041

Test:0.443012263638

Coef:[1.94901771e+00 1.79935890e-01 5.75772252e-02 -8.45991886e-03 3.82283840e-04 -7.06992367e-06 4.63100297e-08]

Degree:7

Train:0.45074188066

Test:0.442575342636

Coef:[1.70847546e+00 3.47096902e-01 1.39022021e-02 -2.95198356e-03 2.49609809e-05 4.72513678e-06 -1.40946836e-07 1.12958340e-09]

Degree:8

Train:0.450231148043

Test:0.442487701726

Coef:[3.93690397e-01 1.40863997e+00 -3.13749111e-01 4.74923395e-02 -4.20781201e-03 2.03296270e-04 -5.28225580e-06 6.93371692e-08 -3.60135053e-10]

Degree:9

Train:0.449716240495

Test:0.439253459889

Coef:[6.57779065e-01 7.66518411e-01 4.06647965e-02 -3.84794380e-02 6.77043490e-03 -5.85586427e-04 2.71796542e-05 -6.83946306e-07 8.76795980e-09 -4.47490401e-11]

Degree:10

Train:0.485239771582

Test:0.457827607951

Coef:[8.10035383e-03 2.95349476e-02 7.80426486e-02 1.25171126e-01 -4.56697918e-02 6.52647339e-03 -4.84697958e-04 2.00717535e-05 -4.64486475e-07 5.59715969e-09 -2.72729507e-11]

Degree:11

Train:0.627028248862

Test:0.646353376223

Coef:[3.13393784e-04 2.12106023e-03 5.06678428e-03 1.51375880e-02 2.59611730e-02 -9.65927443e-03 1.37038921e-03 -1.00082775e-04 4.07376218e-06 -9.28854537e-08 1.10590758e-09 -5.33743373e-12]

Degree:12

Train:2.85408190325

Test:4.53394584743

Coef:[1.20060415e-08 -1.67445314e-06 5.25536588e-07 3.21145550e-06 1.74858508e-05 7.57760222e-05 1.95196988e-04 -5.09012896e-05 4.96757208e-06 -2.38620849e-07 6.03395828e-09 -7.69501605e-11 3.89503838e-13]

Degree:13

Train:8.18203669655

Test:10.0715024042

Coef:[3.95168107e-14 1.06781904e-10 3.10753427e-12 3.01578879e-11 2.67364021e-10 2.24185996e-09 1.68928241e-08 1.03194964e-07 3.87685919e-07 -6.75444710e-08 4.24511502e-09 -1.25235102e-10 1.76192140e-12 -9.53340770e-15]

Degree:14

Train:9.554158671

Test:11.0448045432

Coef:[2.16639158e-16 -5.08056120e-11 -2.29606780e-14 1.97454960e-13 1.89012696e-12 1.75238256e-11 1.52805342e-10 1.19077076e-09 7.48268931e-09 2.87797828e-08 -4.94168737e-09 3.08053486e-10 -9.04048899e-12 1.26734339e-13 -6.83975029e-16]

Degree:15

Train:10.6493250888

Test:11.7727856209

Coef:[1.17714518e-18 -4.45631245e-13 -4.58771397e-15 1.23856013e-15 1.24714430e-14 1.23881558e-13 1.18594897e-12 1.06296521e-11 8.47778730e-11 5.43078453e-10 2.12212634e-09 -3.59970437e-10 2.22832356e-11 -6.50993035e-13 9.09743964e-15 -4.89877188e-17]

Degree:16

Train:14.658067654

Test:14.6303906263

Coef:[2.58346426e-25 -1.12064050e-15 -2.07631352e-19 6.14616717e-22 5.33209390e-21 6.67769289e-20 8.47113777e-19 1.08018529e-17 1.36403547e-16 1.65985013e-15 1.85096217e-14 1.70915296e-13 9.99592835e-13 -9.93492947e-14 3.61772635e-15 -5.74964411e-17 3.36561166e-19]

The testing MSE gets to the minimum point when the degree is 1.

So the best model is:

$review/taste = 2.99503282 + 0.11690802 * beer/ABV$

And the testing MSE is: 0.423776528023

```
train = data[:25000]
test = data[25000:]
def features(datum, degree):
    feat = [1]
    for i in xrange(1, degree+1):
        feat.append(datum['beer/ABV']**i)
    return feat
lastTestingError = 0
deg = 0
while True:
    regr = linear_model.LinearRegression(fit_intercept=False)
    X = [features(d, deg) for d in train]
    y = [d['review/taste'] for d in train]
    X_test = [features(d, deg) for d in test]
    y_test = [d['review/taste'] for d in test]
    regr.fit(X, y)
    thisTestingError = np.mean((regr.predict(X_test) - y_test) ** 2)
    thisTrainError = np.mean((regr.predict(X)-y) ** 2)
    print "Degree:%s\\\\"%(len(X[0])-1)
    print "Train:%s\\\\"% thisTrainError
    print "Test:%s\\\\"% thisTestingError
    print "Coef:%s\\\\"% regr.coef_
    if np.fabs(thisTestingError-lastTestingError) < 0.000001:
        break
    deg = deg + 1
    lastTestingError = thisTestingError
```

Classfication

1

The training accuracy is 0.750 while the testing accuracy is 0.738.

Code Snippets:

```
X = [[1, "child" in s['description'],
         "magic" in s['description'],
         "funny" in s['description'],] for s in data]
y = ["Children's Books" in d['categories'] for d in data]
X_train = X[:len(X)/2]
X_test = X[len(X)/2:]
y_train = y[:len(X)/2]
y_test = y[len(y)/2:]
clf = svm.SVC(C=1000)
clf.fit(X_train, y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
print 'Training Accuracy:%.3f, Testing Accuracy:%.3f'%(1-numpy.mean(
                                        train_predictions!=y_train), 1-numpy.
                                        mean(test_predictions!=y_test))
```

2

The better model's feature vector is ['child', 'magic', 'funny', 'kid', 'dog', 'cat', 'education', 'pat', 'grow']

And the testing error is 0.250, which means testing accuracy is 0.750 which is better than the model in question 1.

```
X = [[1, "child" in s['description'],
         "magic" in s['description'],
         "funny" in s['description'],
         "kid" in s['description'],
         "dog" in s['description'],
```

```
          "cat" in s['description'],
          "education" in s['description'],
          "pat" in s['description'],
          "grow" in s['description']] for s in data]
y = ["Children's Books" in d['categories'] for d in data]
X_train = X[:len(X)/2]
X_test = X[len(X)/2:len(X)]
y_train = y[:len(X)/2]
y_test = y[len(y)/2:len(y)]
clf = svm.SVC(C=1000)
clf.fit(X_train, y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
print 'Training Error:%.3f, Testing Error:%.3f'%(numpy.mean(train_predictions!=
                                          y_train), numpy.mean(test_predictions!=
                                          y_test))
print 'Training Accuracy:%.3f, Testing Accuracy:%.3f'%(1-numpy.mean(
                                          train_predictions!=y_train),1-numpy.mean
                                          (test_predictions!=y_test))
```

3

c=0.001, Train Error:0.492, Valid Error:0.509, Test Error:0.507

c=0.01, Train Error:0.252, Valid Error:0.254, Test Error:0.273

c=0.1, Train Error:0.252, Valid Error:0.254, Test Error:0.273

c=1, Train Error:0.250, Valid Error:0.251, Test Error:0.272

c=10, Train Error:0.250, Valid Error:0.251, Test Error:0.272

c=100, Train Error:0.250, Valid Error:0.251, Test Error:0.272

c=1000, Train Error:0.250, Valid Error:0.251, Test Error:0.272

The test error is going down as the c goes up, and the test error(0.272) of those when c=1, 10, 100, 1000 best reflects the model's ability to generalize to new data.

Code Snippets:

```
X = [[1, "child" in s['description'],
```

```
        "magic" in s['description'],
        "funny" in s['description'],] for s in data]
y = ["Children's Books" in d['categories'] for d in data]
Cs = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 100000]
for c in Cs:
    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:]
    clf = svm.SVC(C=c)
    clf.fit(X_train, y_train)
    train_predictions = clf.predict(X_train)
    valid_predictions = clf.predict(X_valid)
    test_predictions = clf.predict(X_test)
    print 'c=%s, Train Error:%.3f, Valid Error:%.3f, Test Error:%.3f'%\
    (c, numpy.mean(train_predictions != y_train), \
     numpy.mean(valid_predictions != y_valid), \
     numpy.mean(test_predictions != y_test))
```

4

fprime:

```
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])
```

lambda = 0.0001

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

Final log likelihood = -2289.84181645

lambda = 0.001

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

Final log likelihood = -2289.84846705

lambda = 0.01

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

Final log likelihood = -2289.91495803

lambda = 0.1

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

Final log likelihood = -2290.57836569

lambda = 1

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

Final log likelihood = -2297.06998075

lambda = 10

My Train Accuracy:0.748, My Test Accuracy:0.727, My Valid Accuracy:0.746

Final log likelihood = -2351.87280459

lambda = 100

My Train Accuracy:0.748, My Test Accuracy:0.727, My Valid Accuracy:0.746

Final log likelihood = -2589.99994874

lambda = 1000

My Train Accuracy:0.748, My Test Accuracy:0.727, My Valid Accuracy:0.746

Final log likelihood = -2817.8378195

We can see the accuracy(Test, Valid) gets to the maximum point when lambda = 0.0001, 0.001, 0.01, 0.1, 1

The log-likelihood after convergence are: -2289.84181645, -2289.84846705, -2289.91495803, -2290.57836569, -2290.57836569 correspondingly.

The accuracy on test set is: 0.729