preparation

Download MNIST data set and unzip

- http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz (http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz)
- http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz (http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz)
- http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz (http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz)
- http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz (http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz)

Download data loader function

- http://cseweb.ucsd.edu/~dasgupta/dse210/loader.py (http://cseweb.ucsd.edu/~dasgupta/dse210/loader.py)

In [1]:
```
% ls
```

```
 Volume in drive C has no label.
 Volume Serial Number is 3645-AFEC

 Directory of c:\wen\DSE\w9yan\DSE210\HW3

02/18/2017  07:52 PM    <DIR>          .
02/18/2017  07:52 PM    <DIR>          ..
02/16/2017  08:00 PM    <DIR>          .ipynb_checkpoints
02/16/2017  07:56 PM             1,236 loader.py
02/16/2017  08:07 PM             1,349 loader.pyc
02/16/2017  07:57 PM         7,840,016 t10k-images-idx3-ubyte
02/16/2017  07:57 PM            10,008 t10k-labels-idx1-ubyte
02/16/2017  07:57 PM        47,040,016 train-images-idx3-ubyte
02/16/2017  07:57 PM            60,008 train-labels-idx1-ubyte
02/18/2017  07:52 PM            47,860 worksheet6_problem9.ipynb
               7 File(s)     55,000,493 bytes
               3 Dir(s)  131,647,520,768 bytes free
```

In [2]:
```
from struct import unpack
import numpy as np
import matplotlib.pylab as plt
import collections
from __future__ import division
import os.path
from scipy.stats import multivariate_normal
import loader
```

In [3]:
```
# a. load train dataset
training,label = loader.loadmnist('train-images-idx3-ubyte', 'train-labels-idx1-u
print training.shape
print label.shape
```

```
(60000L, 784L)
(60000L,)
```

In [4]:
```
# b. split training data into trainning and validation
training_x, validation_x = training[:50000],training[50000:]
training_y, validation_y = label[:50000],label[50000:]
print training_x.shape, training_y.shape
print validation_x.shape, validation_y.shape
```

```
(50000L, 784L) (50000L,)
(10000L, 784L) (10000L,)
```

In [5]:
```
# b. load test dataset
test_x,test_y = loader.loadmnist('t10k-images-idx3-ubyte', 't10k-labels-idx1-ubyt
test_x.shape, test_y.shape
```

Out[5]:
```
((10000L, 784L), (10000L,))
```

In [6]:
```
# c. calc prior probabilities for each digit
def get_priors(data_labels):
    total = data_labels.shape[0]
    label_count = dict(collections.Counter(data_labels))
    priors = {l:float(count)/total for l,count in label_count.items()}
    return priors

priors = get_priors(training_y)
klasses = priors.keys()
print priors
print sum(priors.values())
```

```
{0: 0.09864, 1: 0.11356, 2: 0.09936, 3: 0.10202, 4: 0.09718, 5: 0.09012, 6: 0.0
9902, 7: 0.1035, 8: 0.09684, 9: 0.09976}
1.0
```

In [7]:
```
# c. calc covariance matrix and posterior probabilities for each class/digit
def get_label_samples(label, data_x, data_y):
    label_samples = [sample for i,sample in enumerate(data_x) if data_y[i] == lab
    return np.matrix(label_samples)

means = {}
cov_matrix = {}
posteriors = []

for klass in klasses:
    samples = get_label_samples(klass, training_x, training_y)
    means[klass] = np.array(samples.mean(0))[0]
    cov_matrix[klass] = np.cov(samples.T)
    posterior = multivariate_normal(allow_singular = True, mean=means[klass], cov
    posteriors.append(posterior)
```

In [8]:
```
# c. define a classify function with priors, posteriors and sample set to classif
def classify(priors, posteriors, sample_set):
    predictions = []
    likelihoods = []
    for x in sample_set:
        likelihood = []
        for klass in priors.keys():
            prob = [klass, np.log(priors[klass]) + posteriors[klass].logpdf(x)]
            likelihood.append(prob)
        likelihoods.append(likelihood)
        prediction = max(likelihood, key=lambda a: a[1])
        predictions.append(prediction[0])
    return predictions, likelihoods
```

In [9]:
```
# calculate error rate on validation set without smoothing covariance matrix
validation_c,likelihoods_c = classify(priors, posteriors, validation_x)
errors = (validation_y != validation_c).sum()
total = len(validation_y)
print('validation error rate: %d/%d = %f' %(errors,total,(errors/float(total))))
```

```
validation error rate: 1840/10000 = 0.184000
```

In [10]:
```
smooth_constant_init = 100000
for klass,cov in cov_matrix.items():
    diagvalues = np.diag(cov)
    nonzeros = [v for v in diagvalues if v != 0]
    meanvalue = sum(nonzeros)/len(nonzeros)
    smooth_constant_init = min(meanvalue, smooth_constant_init)
    print "cov matrix %d diagonal values range: %f ~ %f, mean: %f" % (klass, min(
```

```
cov matrix 0 diagonal values range: 0.007299 ~ 12819.287961, mean: 5792.801168
cov matrix 1 diagonal values range: 0.000176 ~ 13065.216966, mean: 2552.844269
cov matrix 2 diagonal values range: 0.001812 ~ 12976.365114, mean: 5456.409159
cov matrix 3 diagonal values range: 0.000784 ~ 12582.722259, mean: 5069.053553
cov matrix 4 diagonal values range: 0.001852 ~ 12534.168256, mean: 4581.500960
cov matrix 5 diagonal values range: 0.005548 ~ 12594.094403, mean: 5354.354818
cov matrix 6 diagonal values range: 0.016360 ~ 12514.598790, mean: 5054.132936
cov matrix 7 diagonal values range: 0.000773 ~ 12713.347427, mean: 4250.835436
cov matrix 8 diagonal values range: 0.018377 ~ 12339.620571, mean: 5409.221826
cov matrix 9 diagonal values range: 0.009824 ~ 12499.987641, mean: 4628.447648
```

In [11]:
```
# d. smooth the covariance matrices
# I'm going to use same smooth constant for all 10 cov matrix.
# find best smooth constant c with range (0.01, 0.9) * smooth_c_init
errors_v = {}
start = smooth_constant_init * 0.5
end = smooth_constant_init * 1.5
iter_count = 10
step = int((end - start) / iter_count)
for c in np.arange(start,end,step):
    posteriors_v=[]
    for klass in klasses:
        cov = cov_matrix[klass]
        cov_smooth = cov + (c * np.eye(cov.shape[0]))
        p_x = multivariate_normal(allow_singular = True, mean=means[klass], cov=c
        posteriors_v.append(p_x)
    validation_c,likelihoods_c = classify(priors, posteriors_v, validation_x)
    errors_v[c] = (validation_y != validation_c).sum()
```

In [12]:
```
total_v = validation_y.shape[0]
print errors_v
smooth_constant = min(errors_v, key=errors_v.get)
print "Final smooth constant to use is %f" % (smooth_constant)
print "Error rate of validation with this constant is %f" % (errors_v[smooth_cons
```

```
{3316.4221342909373: 410, 2296.4221342909373: 420, 2551.4221342909373: 416, 280
6.4221342909373: 414, 3061.4221342909373: 412, 1786.4221342909373: 435, 1531.42
21342909373: 446, 1276.4221342909373: 456, 3571.4221342909373: 414, 2041.422134
2909373: 426, 3826.4221342909373: 413}
Final smooth constant to use is 3316.422134
Error rate of validation with this constant is 0.041000
```

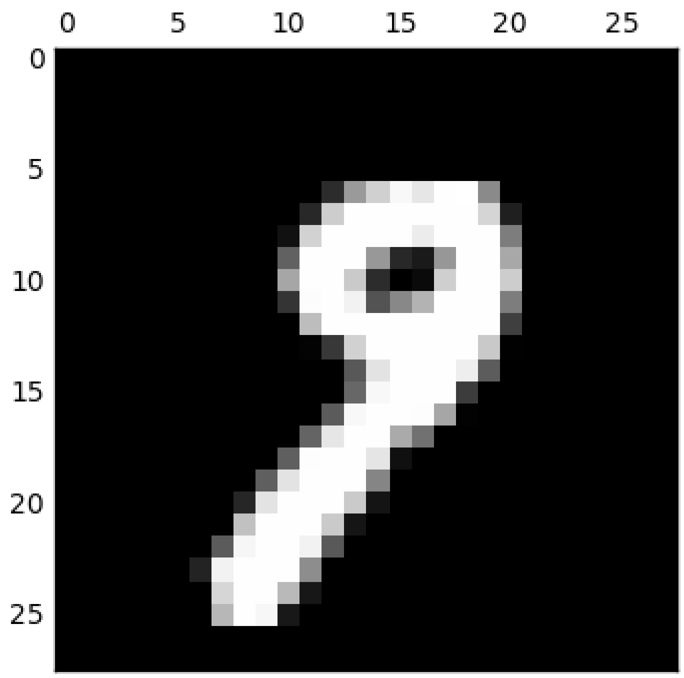
In [13]:
```
# e. build final model with best smooth constant
posteriors_final = []
for klass,cov in cov_matrix.items():
    cov_smooth = cov + (smooth_constant * np.eye(cov.shape[0]))
    mg = multivariate_normal(allow_singular = True, mean=means[klass], cov=cov_sm
    posteriors_final.append(mg)

test_c,likelihoods = classify(priors, posteriors_final, test_x)
errors = (test_y != test_c)
error_rate = float(errors.sum()) / len(test_c)
print "Error rate of test data set is %f" %(error_rate)
```

```
Error rate of test data set is 0.043200
```

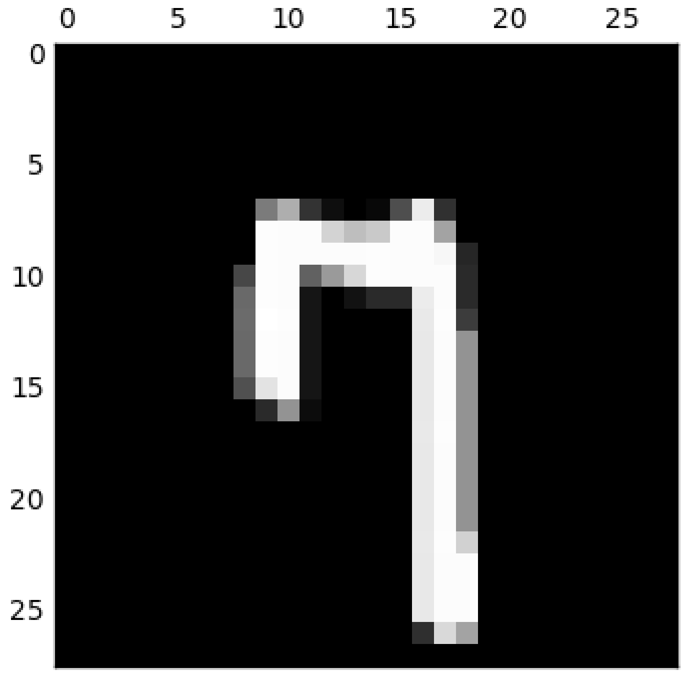
In [14]:

```
error_index = [i for i,x in enumerate(errors.tolist()) if x == 1]
import pylab as pl
pl.gray()
for i in range(5):
    ind = error_index[i]
    img = test_x[ind]
    pl.matshow(np.reshape(img,(28,28)))
    pl.show()
    print "Test label %d vs classified %d" % (test_y[ind], test_c[ind])
    print "The likelihoods for each digit class is:", likelihoods[ind]
```

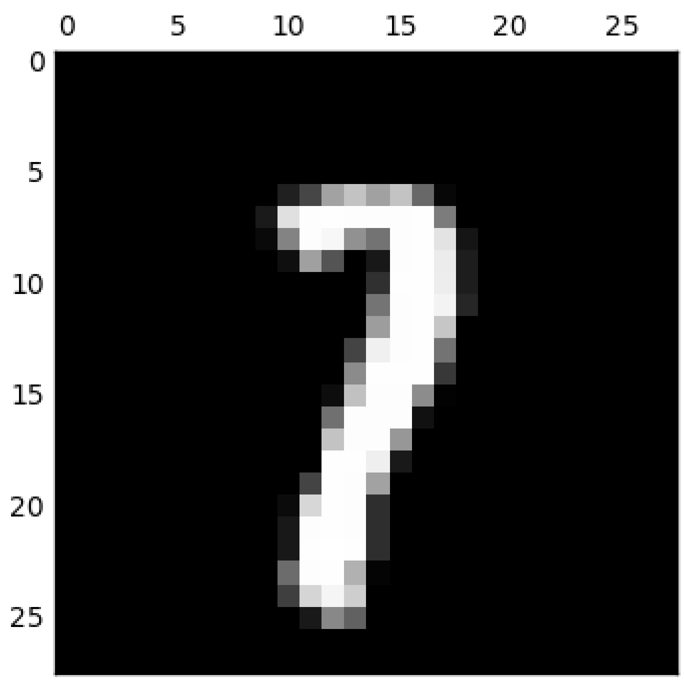
```
<matplotlib.figure.Figure at 0x687f940>
```



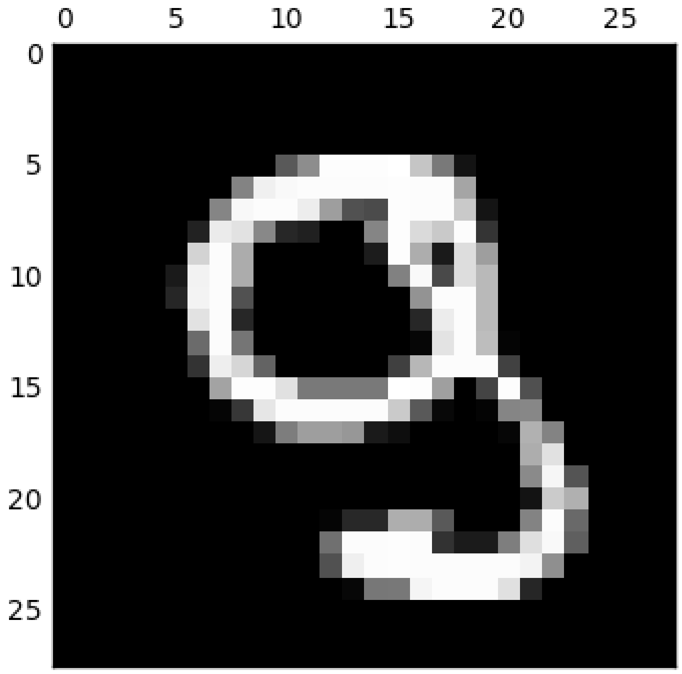

```
Test label 9 vs classified 7
The likelihoods for each digit class is: [[0, -4173.0738073773055], [1, -4141.0
300096695], [2, -4125.8871157210415], [3, -4120.1480398255671], [4, -4115.92388
306472], [5, -4153.8788044507983], [6, -4239.6398436324898], [7, -4051.84673927
01605], [8, -4083.0770631980827], [9, -4057.7084749833598]]
```



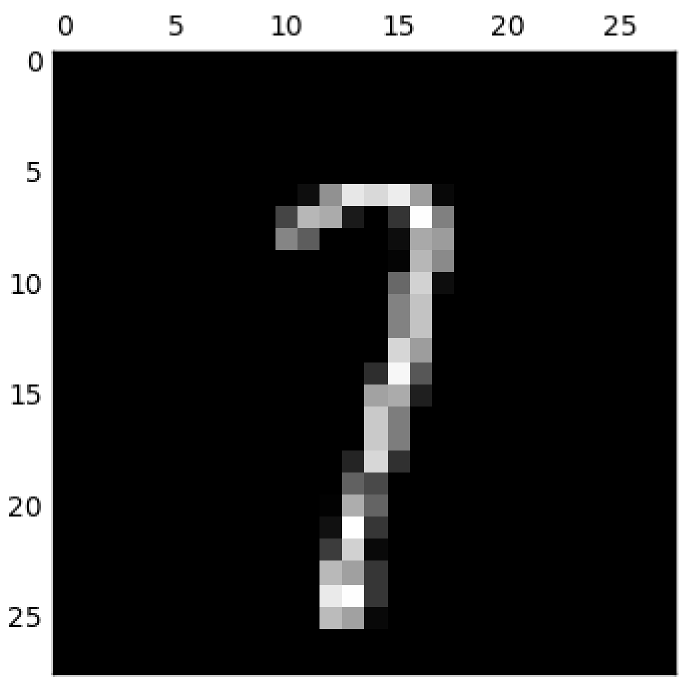
```
Test label 7 vs classified 9
The likelihoods for each digit class is: [[0, -4192.944561094685], [1, -4241.6726543351333], [2, -4177.1717205496616], [3, -4141.2311717225621], [4, -4087.647995254657], [5, -4134.3144360782844], [6, -4245.288248640496], [7, -4050.3912133226218], [8, -4128.0949878232414], [9, -4050.0932792710764]]
```



```
Test label 7 vs classified 1
The likelihoods for each digit class is: [[0, -4146.0256630607455], [1, -4018.9469737308973], [2, -4081.169824345126], [3, -4092.917134771194], [4, -4068.7859165690825], [5, -4126.9575398599727], [6, -4160.6444236432799], [7, -4029.1842680777149], [8, -4060.8515651660687], [9, -4051.00367326586]]
```



Test label 9 vs classified 8
The likelihoods for each digit class is: [[0, -4219.1894035751857], [1, -4479.7258186338449], [2, -4177.6929332903519], [3, -4179.133853403072], [4, -4192.6103225653742], [5, -4178.6472970968462], [6, -4328.6369926823099], [7, -4261.8731601749314], [8, -4146.6259143274319], [9, -4170.4959924941713]]



Test label 7 vs classified 1
The likelihoods for each digit class is: [[0, -4100.8497617049252], [1, -4014.4825701160153], [2, -4083.7705802422206], [3, -4074.6459594051535], [4, -4052.9281028048499], [5, -4088.8902930665622], [6, -4104.8697101156276], [7, -4025.3503461196347], [8, -4065.0332402728495], [9, -4027.527212488922]]