DSE210 Worksheet 6

Orysya Stus

1. $\pi(h) = \frac{3}{4}$ given h $Pr(a\ lot|h) = \frac{2}{3}$ $Pr(little|h) = \frac{1}{6}$ $Pr(silent|h) = \frac{1}{6}$

$\pi(s) = \frac{1}{4}$ given s $Pr(a\ lot|s) = \frac{1}{6}$ $Pr(little|s) = \frac{1}{6}$ $Pr(silent|s) = \frac{2}{3}$

a. $P(s|little) = \frac{P(little|s)P(s)}{P(little)} = \frac{P(little|s)P(s)}{P(little|s)P(s) + P(little|h)P(h)} = \frac{(1/6)(1/4)}{(1/6)(1/4)+(1/6)(3/4)} = \frac{1}{4}$

$P(h|little) = \frac{P(little|h)P(h)}{P(little)} = \frac{(1/6)(3/4)}{4/24} = \frac{3}{4}$

↳ he is most likely happy

b. IP of prediction in part (a) being incorrect

$P(s|little) = \frac{1}{4}$

2. $X = [-1, 1]$ $Y = \{1,2,3\}$ $\pi_1 = \frac{1}{3}$ $\pi_2 = \frac{1}{6}$ $\pi_3 = \frac{1}{2}$ P_1, P_2, P_3 given

$h^*(x) = \arg\max_j \pi_j P_j(x) = \begin{cases} 1 & \text{if } -1 \leq x \leq 0 \\ 3 & \text{if } 0 < x < 1 \end{cases}$

3a. positively correlated

b. positively correlated

c. negatively correlated

4. corr (wife age, husband age) = 1

5. give the parameters of the (unique) bivariate Gaussian that satisfies these props

a. $\mu_x = 2$ $std(x) = 1$ $\mu_y = 2$ $std(y) = 0.5$ $corr(x,y) = -0.5$

$corr(x,y) = \frac{cov(X,Y)}{std(x)std(y)}$ $-0.5 = \frac{cov(X,Y)}{1 \cdot 0.5}$ $cov(X,Y) = -0.25$

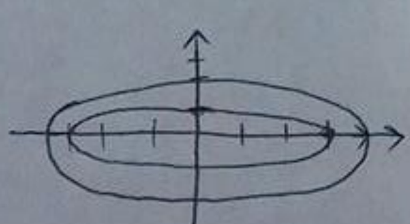
covariance matrix $\Sigma = \begin{pmatrix} var(X) & cov(X,Y) \\ cov(X,Y) & var(Y) \end{pmatrix} = \begin{pmatrix} 1 & -0.25 \\ -0.25 & 0.25 \end{pmatrix}$ $\mu = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$

b. $\mu_x = 1 = \mu_y$ $std(x) = std(y) = 1$ $\mu = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $cov(X,Y) = 1$ covariance matrix $\Sigma = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

6. sketch shapes of Gaussians $N(\mu, \Sigma)$

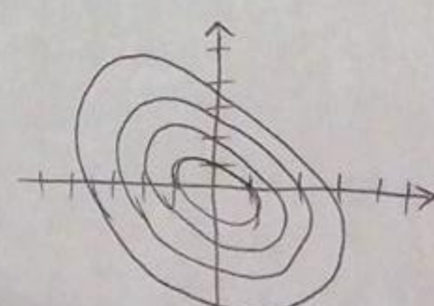
a. $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$\Sigma = \begin{pmatrix} 9 & 0 \\ 0 & 1 \end{pmatrix}$



b. $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$\Sigma = \begin{pmatrix} 1 & -0.75 \\ -0.75 & 1 \end{pmatrix}$



Worksheet 6

February 18, 2017

1 Worksheet 6 – Generative models 2

1.1 Orysya Stus

1.2 2.19.2017

https://github.com/mas-dse/ostus/tree/master/DSE210

```
In [1]: import pandas as pd
        import numpy as np
        import math
        import sklearn
        from sklearn.naive_bayes import MultinomialNB
        from sklearn import metrics
        from sklearn import datasets
        from scipy.stats import multivariate_normal
        import random
        import seaborn as sns
        import matplotlib.pyplot as plt
        import re

        %matplotlib inline
```

```
C:\Users\Orysya\Anaconda\lib\site-packages\IPython\html.py:14: ShimWarning: The `IF
  "`IPython.html.widgets` has moved to `ipywidgets`.", ShimWarning)
```

1.3 Problem 7

For each of the two Gaussians in the previous problem, check your answer using Python: draw 100 random samples from that Gaussian and plot it.

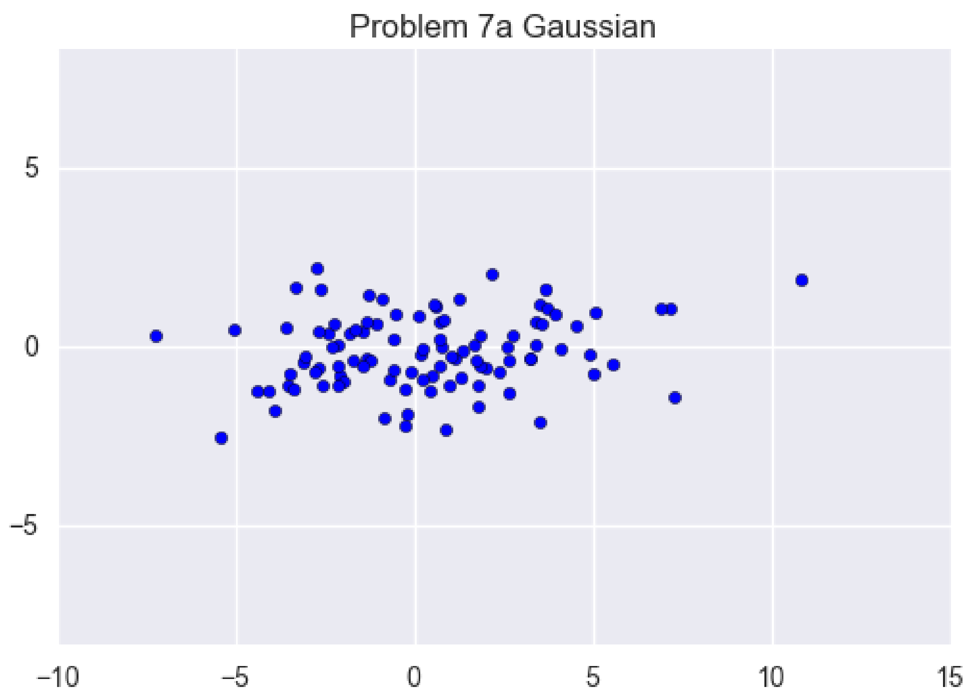
Using: https://docs.scipy.org/doc/numpy/reference/generated/numpy.random.multivariate_normal.html

1.3.1 7a

```
In [2]: mean = [0,0]
        cov = [[9,0], [0,1]]
        x, y = np.random.multivariate_normal(mean, cov, 100).T
        plt.scatter(x, y)
```

```
plt.axis('equal')
plt.title('Problem 7a Gaussian')
```

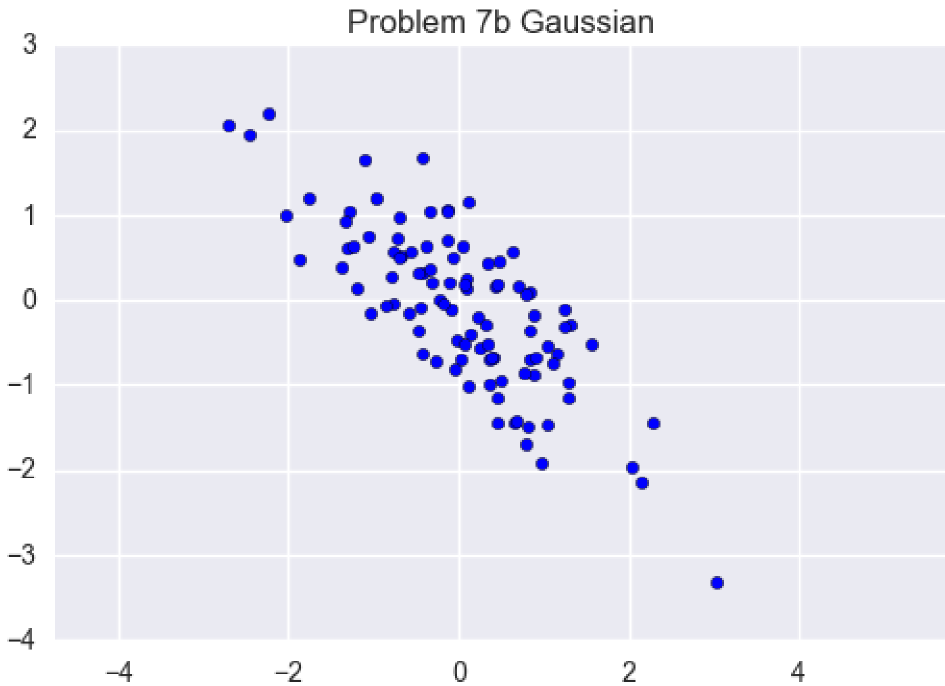
Out[2]: <matplotlib.text.Text at 0xbee0550>



1.3.2 7b

In [3]:
```
mean = [0,0]
cov = [[1, -0.75], [-0.75, 1]]
x, y = np.random.multivariate_normal(mean, cov, 100).T
plt.scatter(x, y)
plt.axis('equal')
plt.title('Problem 7b Gaussian')
```

Out[3]: <matplotlib.text.Text at 0xc0627b8>



1.4 Problem 8

Consider the linear classifer. Sketch the decision boundary in R^2. Make sure to label precisely where the boundary intersects the coordinate axes, and also indicate which side of the boundary is the positive side.

```
In [4]: x = np.linspace(-10, 4, 10, endpoint=True)
        y = (3 * x + 12) / 4
        fig = plt.figure()
        ax = fig.add_subplot(111)
        ax.plot(x, y)
        ax.spines['right'].set_color('none')
        ax.spines['top'].set_color('none')
        ax.xaxis.set_ticks_position('bottom')
        ax.spines['bottom'].set_position(('data',0))
        ax.yaxis.set_ticks_position('left')
        plt.annotate(r'positive', xy=(-6, 2), xycoords='data', xytext=(+50, +30), t

Out[4]: <matplotlib.text.Annotation at 0xc355080>
```

2 Problem 9: Handwritten digit recognition using a Gaussian generative model

Handwritten digit recognition using a Gaussian generative model. In class, we mentioned the MNIST data set of handwritten digits. You can obtain it from: http://yann.lecun.com/exdb/mnist/index.html

In this problem, you will build a classifier for this data, by modeling each class as a multivariate (784-dimensional) Gaussian.

Refer to the following link for review:

http://www.eggie5.com/68-mnist-gaussian-classifier

2.1 Part (a)

Upon downloading the data, you should have two training files (one with imahes, one with labels) and two test files. Unzip them.

In order to load the data into Python you will find the following code helpful:

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

For instance, to load in the training data, you can use:

```
x,y = loadmnist('train-images-idx3-ubyte', 'train-labels-idx1-ubyte')
```

This will set x to a 60000 x 784 array where each row corresponds to an image, and y to a length-60000 array where each entry is a label (0-9). There is also a routine to display images: use displaychar(x[0]) to show the first data point, for instance.

```
In [5]: import loader as loader
        x, y = loader.loadmnist('train-images.idx3-ubyte', 'train-labels.idx1-ubyte
        print 'The shape of x is:', x.shape
        print 'The shape of y is:', y.shape
```

```
The shape of x is: (60000L, 784L)
The shape of y is: (60000L,)
```

2.1.1 Examine the digit data (as an array and image)

```
In [6]: print x[1]
        loader.displaychar(x[1])
```

```
[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  51 159 253 159  50   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0  48 238 252 252 252 237   0   0
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  54 227 253 252 239 233 252  57   6   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0  10  60 224 252 253 252 202  84 252
  253 122   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0 163 252 252 252 253 252 252  96 189 253 167   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0  51 238 253 253 190 114 253 228
   47  79 255 168   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0  48 238 252 252 179  12  75 121  21   0   0 253 243  50   0   0   0
    0   0   0   0   0   0   0   0   0   0  38 165 253 233 208  84   0   0
    0   0   0   0 253 252 165   0   0   0   0   0   0   0   0   0   0   0
    0   7 178 252 240  71  19  28   0   0   0   0   0   0 253 252 195   0
    0   0   0   0   0   0   0   0   0   0   0  57 252 252  63   0   0   0
    0   0   0   0   0   0 253 252 195   0   0   0   0   0   0   0   0   0
    0   0   0 198 253 190   0   0   0   0   0   0   0   0   0   0 255 253
  196   0   0   0   0   0   0   0   0   0   0   0  76 246 252 112   0   0
    0   0   0   0   0   0   0   0 253 252 148   0   0   0   0   0   0   0
    0   0   0   0  85 252 230  25   0   0   0   0   0   0   0   0   7 135
  253 186  12   0   0   0   0   0   0   0   0   0   0   0  85 252 223   0
    0   0   0   0   0   0   0   7 131 252 225  71   0   0   0   0   0   0
    0   0   0   0   0   0  85 252 145   0   0   0   0   0   0   0  48 165
  252 173   0   0   0   0   0   0   0   0   0   0   0   0   0   0  86 253
  225   0   0   0   0   0   0 114 238 253 162   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0  85 252 249 146  48  29  85 178 225 253
  223 167  56   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   85 252 252 252 229 215 252 252 252 196 130   0   0   0   0   0   0   0
```

```
   0    0    0    0    0    0    0    0    0    0   28  199  252  252  253  252  252  233
 145    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0   25  128  252  253  252  141   37    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
   0    0    0    0    0    0    0    0    0    0]
```

2.2 Part (b)

Split the training set into two pieces - a training set of size 50000, and a separate validation set of size 10000. Also load in the test data.

```
In [7]: from sklearn.cross_validation import train_test_split
        X_train, X_validation, y_train, y_validation = train_test_split(x, y, test_
        print 'The shape of the x train data set is: ', X_train.shape
        print 'The shape of the y train data set is: ', y_train.shape
        print 'The shape of the x validation data set is: ', X_validation.shape
        print 'The shape of the y validation data set is: ', y_validation.shape
```

```
The shape of the x train data set is:  (50000L, 784L)
The shape of the y train data set is:  (50000L,)
```

```
The shape of the x validation data set is:  (10000L, 784L)
The shape of the y validation data set is:  (10000L,)


C:\Users\Orysya\Anaconda\lib\site-packages\sklearn\cross_validation.py:44: Deprecat
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [8]: X_test, y_test = loader.loadmnist('t10k-images.idx3-ubyte', 't10k-labels.id
        print 'The shape of the x test data set is: ', X_test.shape
        print 'The shape of the y test data set is: ', y_test.shape
```

```
The shape of the x test data set is:  (10000L, 784L)
The shape of the y test data set is:  (10000L,)
```

2.3 Part (c)

Now fit a Gaussian generative model to the training data of 50000 points:

```
Determine the class probabilities: what fraction of pi_0 of the training points are

Fit a Gaussian to each digit, by finding the mean and the covariance of the corresp
Let the Gaussian for the jth digit by Pj = N(mu_j, sigma_j).
```

Using these two pieces of information, you can classify new images x using Bayes' rule: simply pick the digit j for which pi_j,Pj(x) is largest.

```
In [9]: from sklearn.naive_bayes import GaussianNB
        clf = GaussianNB()
        clf.fit(X_train, y_train)

Out[9]: GaussianNB(priors=None)
```

2.3.1 Show the fraction (percentage) of images that belong to each class. Associated withe PI_j or the prior probabilities. The class distribution is fairly uniform.

```
In [10]: priors = clf.class_prior_
         length = len(priors)
         for i in range(length):
             percentage = (priors[i])
             print 'The prior probability of class image %s is' %i, '{0:.4f}'.forma
```

```
The prior probability of class image 0 is 0.0994
The prior probability of class image 1 is 0.1118
The prior probability of class image 2 is 0.0996
The prior probability of class image 3 is 0.1020
The prior probability of class image 4 is 0.0972
The prior probability of class image 5 is 0.0901
```

```
The prior probability of class image 6 is 0.0981
The prior probability of class image 7 is 0.1042
The prior probability of class image 8 is 0.0978
The prior probability of class image 9 is 0.0997
```

2.3.2 Fit the train data set to the model used above (note: the validation set was not yet used/considered)

```
In [11]: classes = clf.classes_
         posteriors = []

         def class_grouping(class_id):
             grouping = []
             for i, group in enumerate(X_train):
                 if y_train[i] == class_id:
                     grouping.append(group)
             grouping = np.matrix(grouping)
             return grouping

         posteriors = []
         for c in classes:
             grouping = class_grouping(c)
             mean = np.array(grouping.mean(0))[0]
             cov = np.cov(grouping.T)
             Px = multivariate_normal(mean, cov, allow_singular=True)
             posteriors.append(Px)
```

2.3.3 Examining model predictions based on Bayes probability

```
In [12]: x = random.choice(X_test)
         actual = y_test[x]

         bayes_prob = []
         for c in classes:
             prob = [c, priors[c] * posteriors[c].pdf(x)]
             formatting_function = np.vectorize(lambda f: format(f, '6.3E'))
             bayes_prob.append(prob)

         print 'The Bayes probability found is \n', bayes_prob
         prediction = max(bayes_prob, key= lambda a: a[1])
```

```
The Bayes probability found is
[[0, 0.0], [1, 0.0], [2, 0.0], [3, 0.0], [4, 0.0], [5, 0.0], [6, 0.0], [7, 0.0], [8
```

2.3.4 Building the classifier

```
In [13]: Y = []
         for x in X_test:
             bayes_prob = []
             for c in classes:
                 prob = [c, priors[c] * posteriors[c].pdf(x)]
                 bayes_prob.append(prob)
             prediction = max(bayes_prob, key= lambda a: a[1])
             Y.append(prediction[0])
```

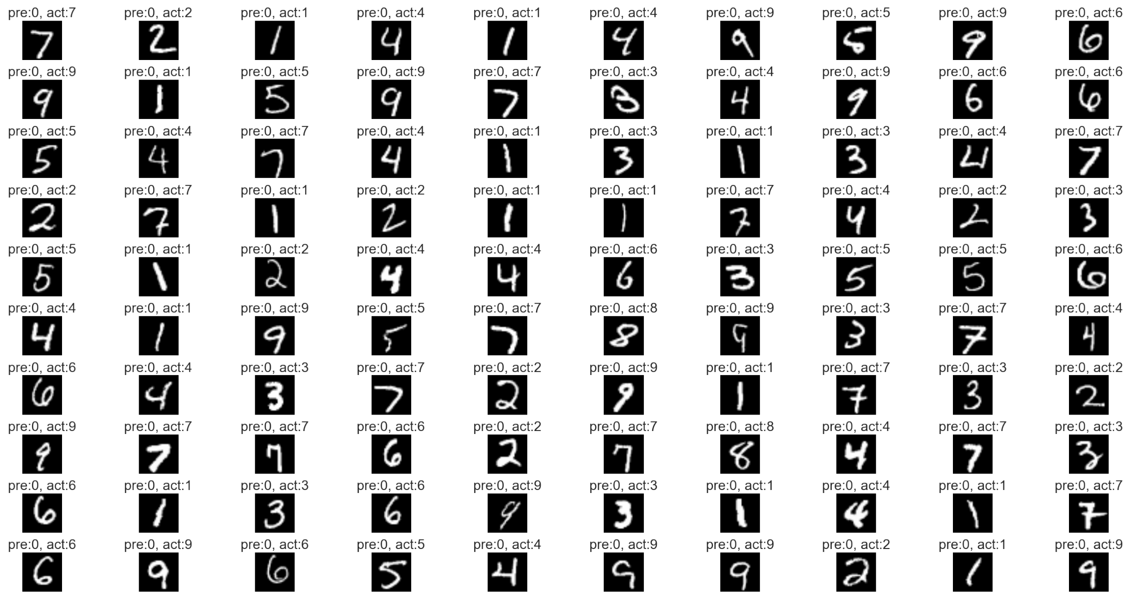
```
In [14]: errors = (y_test != Y).sum()
         total = X_test.shape[0]
         print("Error rate: %d/%d = %f" %((errors, total, (errors/float(total)))))
```

```
Error rate: 9020/10000 = 0.902000
```

2.3.5 Our naively implemented Gaussian Classifer achieved a 9.8% success rate. Note such a low success rate can be due to using 'pdf' vs 'logpdf' which results in very small probabilities. Loss in probability precision results in inaccurate predictions.

2.3.6 Comparing 100 of the predicted and actual classes for the digit data, the model predicted all of the digits to be 0s.

```
In [15]: def displayimage(image):
             plt.imshow(np.reshape(image, (28, 28)), cmap=plt.cm.gray)
             plt.axis('off')

         indices = np.array(np.where((y_test != Y)==True))[0]
         indices = indices[0:100]
         index = 0
         rows = len(indices)/10
         cols = 10

         plt.figure(figsize=(30,15))
         for i in indices:
             index += 1
             plt.subplot(rows, cols, index)
             plt.subplots_adjust(hspace=.5)
             displayimage(X_test[i])
             plt.title('pre:%i, act:%i' %( Y[i], y_test[i]), fontsize = 20)
```



2.3.7 Building the classifier: Using log probabilities

```
In [16]: Y = []
         for x in X_test:
             bayes_prob = []
             for c in classes:
                 prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
                 bayes_prob.append(prob)
             prediction = max(bayes_prob, key= lambda a: a[1])
             Y.append(prediction[0])
```

```
In [17]: errors = (y_test != Y).sum()
         total = X_test.shape[0]
         print("Error rate: %d/%d = %f" %((errors, total, (errors/float(total)))))
```

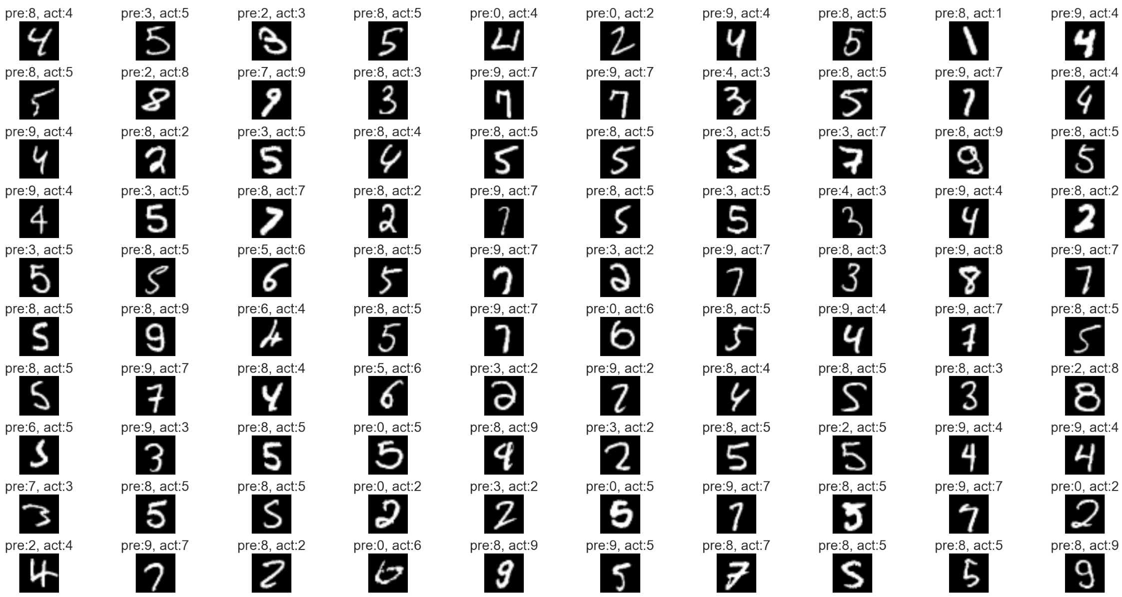
```
Error rate: 1862/10000 = 0.186200
```

2.3.8 Our naively implemented Gaussian Classifer achieved a 81.4% success rate. Using log-pdf resulted in more precise probabilities, thus a more accurate model.

2.3.9 Comparing 100 of the predicted and actual classes for the digit data, the model predicted all of the digits to be 0s.

```
In [18]: indices = np.array(np.where((y_test != Y)==True))[0]
         indices = indices[0:100]
         index = 0
         rows = len(indices)/10
```

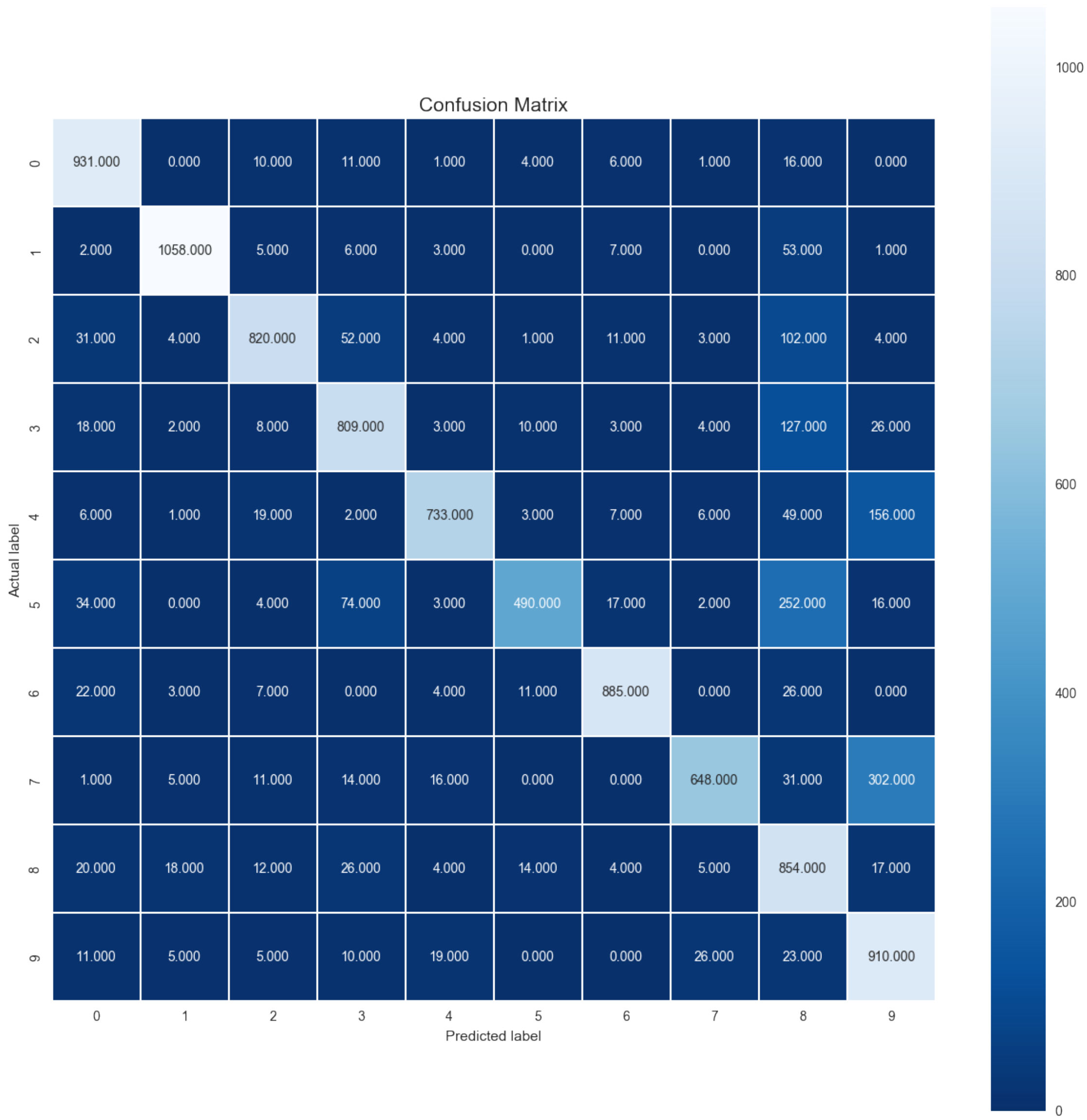
```
cols = 10

plt.figure(figsize=(30,15))
for i in indices:
    index += 1
    plt.subplot(rows, cols, index)
    plt.subplots_adjust(hspace=.5)
    displayimage(X_test[i])
    plt.title('pre:%i, act:%i' %( Y[i], y_test[i]), fontsize = 20)
```




```
In [19]: print 'The model (no validation set used) has an accuracy of', metrics.acc
         print metrics.classification_report(y_test, Y)

         cm = pd.DataFrame(metrics.confusion_matrix(y_test, Y))
         plt.figure(figsize=(15, 15))
         sns.heatmap(cm, annot=True, fmt=".3f", linewidth=.5, square = True, cmap =
         plt.ylabel('Actual label');
         plt.xlabel('Predicted label');
         plt.title('Confusion Matrix', size = 15);
```

```
The model (no validation set used) has an accuracy of 0.8138
             precision    recall  f1-score   support

          0       0.87      0.95      0.91       980
          1       0.97      0.93      0.95      1135
          2       0.91      0.79      0.85      1032
          3       0.81      0.80      0.80      1010
          4       0.93      0.75      0.83       982
```

5	0.92	0.55	0.69	892
6	0.94	0.92	0.93	958
7	0.93	0.63	0.75	1028
8	0.56	0.88	0.68	974
9	0.64	0.90	0.75	1009
avg / total	0.85	0.81	0.82	10000



2.3.10 The model above misclassified digits 5 & 7 most often.

2.4 Part (d)

One last step is needed: it is important to smooth the covariance matrices, and the usual way to do this is to add in cI, where c is some constant and I is the identity matrix. What value of c is right? Use the validation set to help you choose. That is, choose the value of c for which the resulting classifier makes the fewest mistakes on the validation set. What value of c did you get?

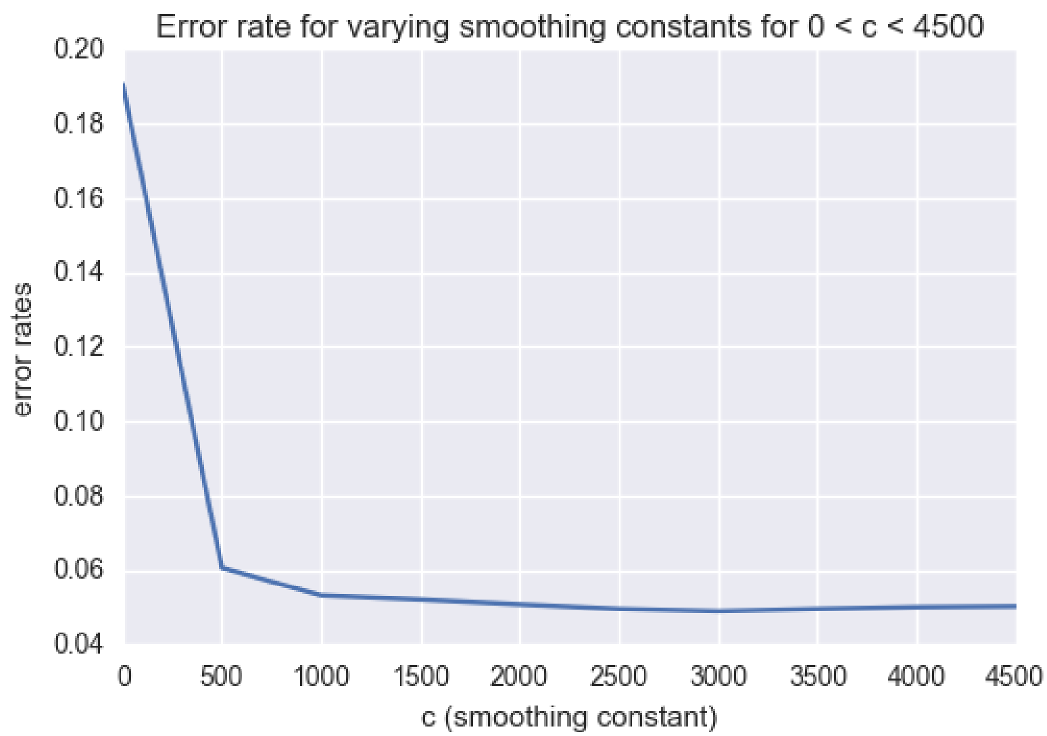
```
In [20]: smoothing_c = range(0, 5000, 500)
         error_rates = []
         for sc in smoothing_c:
             posteriors = []
             for c in classes:
                 grouping = class_grouping(c)
                 mean = np.array(grouping.mean(0))[0]
                 cov = np.cov(grouping, rowvar=0)
                 cov_smoothed = cov + (sc * np.eye(mean.shape[0]))
                 p_x = multivariate_normal(mean, cov_smoothed, allow_singular=True)
                 posteriors.append(p_x)

             Y = []
             for x in X_validation:
                 bayes_prob = []
                 for c in classes:
                     prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
                     bayes_prob.append(prob)
                 prediction = max(bayes_prob, key= lambda a: a[1])
                 Y.append(prediction[0])

             errors = (y_validation != Y).sum()
             total = X_validation.shape[0]
             error_rate = errors/float(total)
             error_rates.append(error_rate)
             print("Error rate for c= %s: %d/%d = %f" %((sc, errors, total, error_r
```

```
Error rate for c= 0: 1906/10000 = 0.190600
Error rate for c= 500: 606/10000 = 0.060600
Error rate for c= 1000: 532/10000 = 0.053200
Error rate for c= 1500: 521/10000 = 0.052100
Error rate for c= 2000: 508/10000 = 0.050800
Error rate for c= 2500: 496/10000 = 0.049600
Error rate for c= 3000: 490/10000 = 0.049000
Error rate for c= 3500: 496/10000 = 0.049600
Error rate for c= 4000: 501/10000 = 0.050100
Error rate for c= 4500: 503/10000 = 0.050300
```

```
In [21]: plt.plot(smoothing_c, error_rates)
         plt.xlabel('c (smoothing constant)')
```

```
plt.ylabel('error rates')
plt.title('Error rate for varying smoothing constants for 0 < c < 4500')
```

```
Out[21]: <matplotlib.text.Text at 0x220280f0>
```



2.4.1 The optimal c lies within 2500 < smoothing_c < 3500, iterative smoothing is run again to determine the optimal smoothing c.

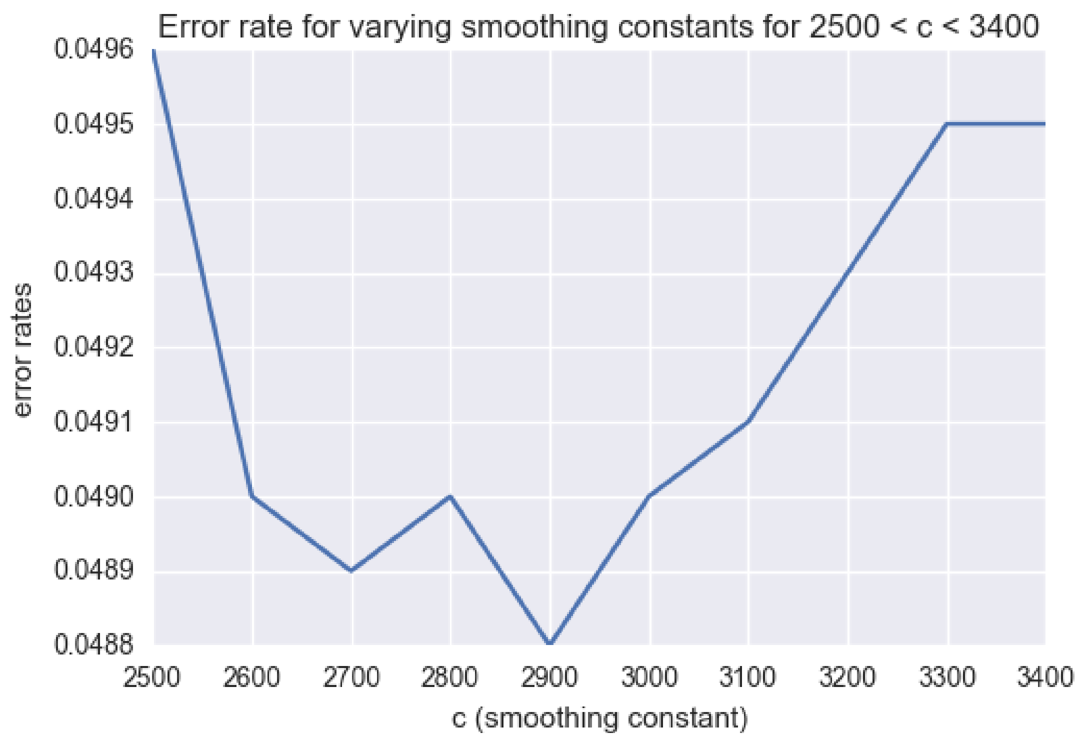
```
In [22]: smoothing_c = range(2500, 3500, 100)
         error_rates = []
         for sc in smoothing_c:
             posteriors = []
             for c in classes:
                 grouping = class_grouping(c)
                 mean = np.array(grouping.mean(0))[0]
                 cov = np.cov(grouping, rowvar=0)
                 cov_smoothed = cov + (sc * np.eye(mean.shape[0]))
                 p_x = multivariate_normal(mean, cov_smoothed, allow_singular=True)
                 posteriors.append(p_x)

             Y = []
             for x in X_validation:
                 bayes_prob = []
```

```
                for c in classes:
                    prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
                    bayes_prob.append(prob)
                prediction = max(bayes_prob, key= lambda a: a[1])
                Y.append(prediction[0])

            errors = (y_validation != Y).sum()
            total = X_validation.shape[0]
            error_rate = errors/float(total)
            error_rates.append(error_rate)
            print("Error rate for c= %s: %d/%d = %f" %((sc, errors, total, error_r
```

```
Error rate for c= 2500: 496/10000 = 0.049600
Error rate for c= 2600: 490/10000 = 0.049000
Error rate for c= 2700: 489/10000 = 0.048900
Error rate for c= 2800: 490/10000 = 0.049000
Error rate for c= 2900: 488/10000 = 0.048800
Error rate for c= 3000: 490/10000 = 0.049000
Error rate for c= 3100: 491/10000 = 0.049100
Error rate for c= 3200: 493/10000 = 0.049300
Error rate for c= 3300: 495/10000 = 0.049500
Error rate for c= 3400: 495/10000 = 0.049500
```

In [23]:
```
plt.plot(smoothing_c, error_rates)
plt.xlabel('c (smoothing constant)')
plt.ylabel('error rates')
plt.title('Error rate for varying smoothing constants for 2500 < c < 3400'
```

Out[23]:
```
<matplotlib.text.Text at 0x200b4d30>
```



2.4.2 The optimal c lies within 2800 < smoothing_c < 3000, iterative smoothing is run again to determine the optimal smoothing c.

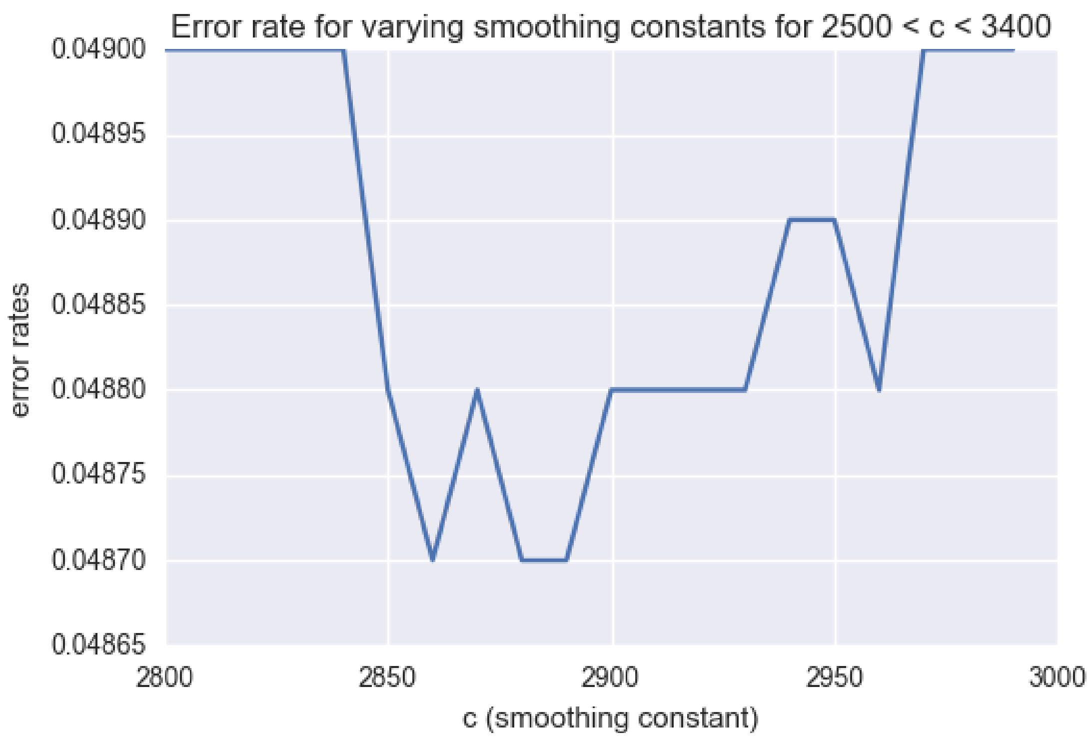
```
In [24]: smoothing_c = range(2800, 3000, 10)
         error_rates = []
         for sc in smoothing_c:
             posteriors = []
             for c in classes:
                 grouping = class_grouping(c)
                 mean = np.array(grouping.mean(0))[0]
                 cov = np.cov(grouping, rowvar=0)
                 cov_smoothed = cov + (sc * np.eye(mean.shape[0]))
                 p_x = multivariate_normal(mean, cov_smoothed, allow_singular=True)
                 posteriors.append(p_x)

             Y = []
             for x in X_validation:
                 bayes_prob = []
                 for c in classes:
                     prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
                     bayes_prob.append(prob)
                 prediction = max(bayes_prob, key= lambda a: a[1])
                 Y.append(prediction[0])
```

```
            errors = (y_validation != Y).sum()
            total = X_validation.shape[0]
            error_rate = errors/float(total)
            error_rates.append(error_rate)
            print("Error rate for c= %s: %d/%d = %f" %((sc, errors, total, error_r
```

```
Error rate for c= 2800: 490/10000 = 0.049000
Error rate for c= 2810: 490/10000 = 0.049000
Error rate for c= 2820: 490/10000 = 0.049000
Error rate for c= 2830: 490/10000 = 0.049000
Error rate for c= 2840: 490/10000 = 0.049000
Error rate for c= 2850: 488/10000 = 0.048800
Error rate for c= 2860: 487/10000 = 0.048700
Error rate for c= 2870: 488/10000 = 0.048800
Error rate for c= 2880: 487/10000 = 0.048700
Error rate for c= 2890: 487/10000 = 0.048700
Error rate for c= 2900: 488/10000 = 0.048800
Error rate for c= 2910: 488/10000 = 0.048800
Error rate for c= 2920: 488/10000 = 0.048800
Error rate for c= 2930: 488/10000 = 0.048800
Error rate for c= 2940: 489/10000 = 0.048900
Error rate for c= 2950: 489/10000 = 0.048900
Error rate for c= 2960: 488/10000 = 0.048800
Error rate for c= 2970: 490/10000 = 0.049000
Error rate for c= 2980: 490/10000 = 0.049000
Error rate for c= 2990: 490/10000 = 0.049000
```

```
In [25]: plt.plot(smoothing_c, error_rates)
         plt.xlabel('c (smoothing constant)')
         plt.ylabel('error rates')
         plt.title('Error rate for varying smoothing constants for 2500 < c < 3400'
```

```
Out[25]: <matplotlib.text.Text at 0x1128fc88>
```



2.4.3 Using the validation data set, a couple of candidates for the optimal smoothing_c were found:

Error rate for c= 2860: 487/10000 = 0.048700
Error rate for c= 2880: 487/10000 = 0.048700
Error rate for c= 2890: 487/10000 = 0.048700

2.4.4 Smoothing_c = 2860 was choosen, which yielded an error rate of 4.87% using the validation data set.

2.5 Part (e)

Turn in an iPython that includes:

```
All your code
Error rate on the MNIST test set
Out of the misclassified test digits, pick 5 at random and display them. For each i
```

```
In [26]: sc = 2860
         posteriors = []
         for c in classes:
             grouping = class_grouping(c)
             mean = np.array(grouping.mean(0))[0]
             cov = np.cov(grouping, rowvar=0)
```

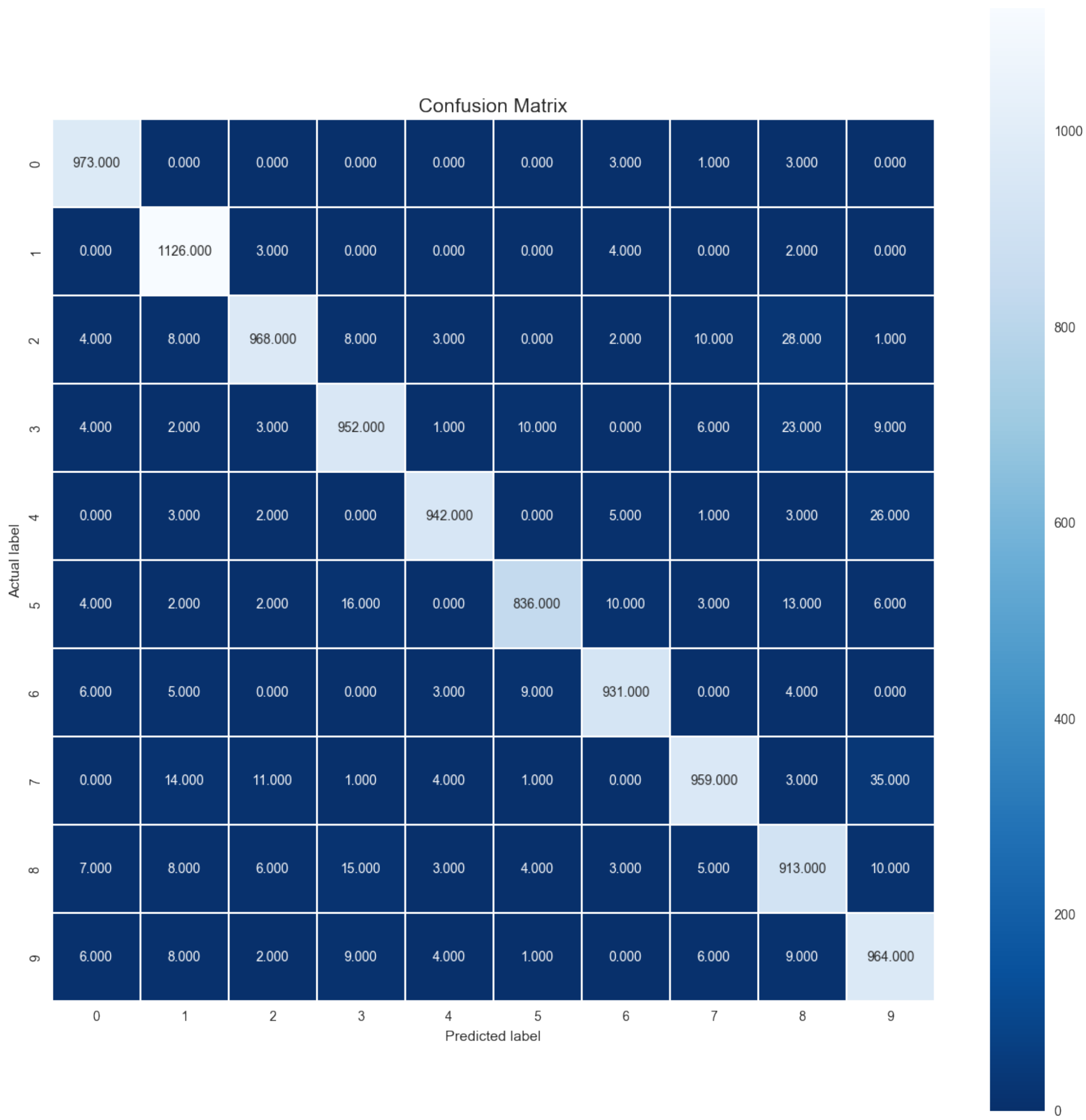
```
        cov_smoothed = cov + (sc * np.eye(mean.shape[0]))
        Px = multivariate_normal(mean, cov_smoothed)
        posteriors.append(Px)

    Y = []
    for x in X_test:
        bayes_prob = []
        for c in classes:
            prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
            bayes_prob.append(prob)
        prediction = max(bayes_prob, key= lambda a: a[1])
        Y.append(prediction[0])

    errors = (y_test != Y).sum()
    total = X_test.shape[0]
    error_rate = errors/float(total)
    print("Error rate for c= %s: %d/%d = %f" %((sc, errors, total, error_rate)
```

```
Error rate for c= 2860: 436/10000 = 0.043600
```

2.6 The error rate on the MNIST test set is: 4.36%

```
In [27]: print 'The model (no validation set used) has an accuracy of', metrics.acc
         print metrics.classification_report(y_test, Y)

         cm = pd.DataFrame(metrics.confusion_matrix(y_test, Y))
         plt.figure(figsize=(15, 15))
         sns.heatmap(cm, annot=True, fmt=".3f", linewidth=.5, square = True, cmap =
         plt.ylabel('Actual label');
         plt.xlabel('Predicted label');
         plt.title('Confusion Matrix', size = 15);
```

```
The model (no validation set used) has an accuracy of 0.9564
             precision    recall  f1-score   support

          0       0.97      0.99      0.98       980
          1       0.96      0.99      0.97      1135
          2       0.97      0.94      0.95      1032
          3       0.95      0.94      0.95      1010
          4       0.98      0.96      0.97       982
          5       0.97      0.94      0.95       892
          6       0.97      0.97      0.97       958
          7       0.97      0.93      0.95      1028
          8       0.91      0.94      0.92       974
          9       0.92      0.96      0.94      1009

avg / total       0.96      0.96      0.96     10000
```

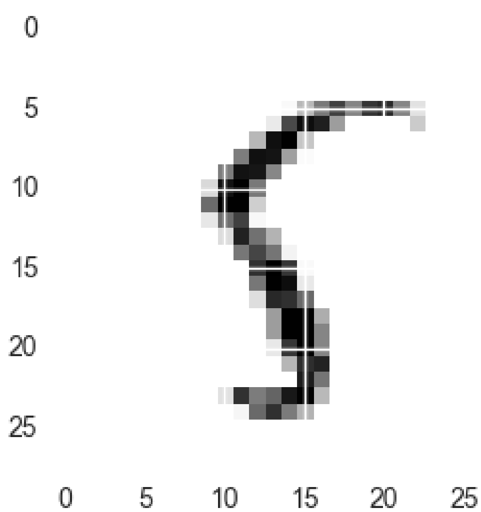


2.6.1 The model was able to correctly predict the majority of the diagonals, with an accuracy of 95.64%. Digits 8 & 9 had the lowest precision (but still > 0.90).

```
In [28]: indices = np.array(np.where((y_test != Y)==True))[0]

         wrong = random.sample(indices, 5)
         actuals = y_test[wrong]
         misclassified_predictions = []
         for x in X_test[wrong]:
```

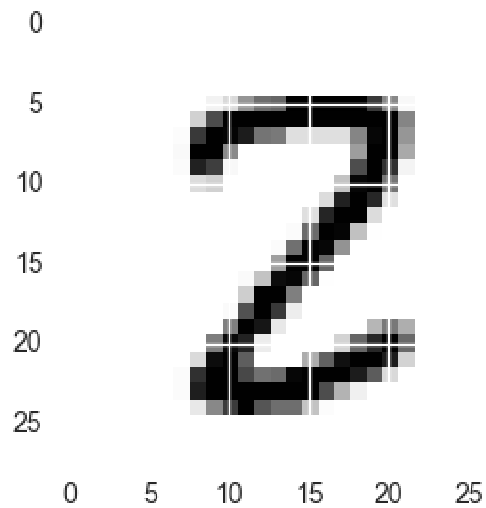
```
            bayes_prob = []
            for c in classes:
                prob = [c, np.log(priors[c]) + posteriors[c].logpdf(x)]
                bayes_prob.append(prob)
            prediction = max(bayes_prob, key= lambda a: a[1])
            misclassified_predictions.append(prediction[0])

        for i in range(len(wrong)):
            bayes_prob = []
            for c in classes:
                prob = [c, np.log(priors[c]) + posteriors[c].logpdf(X_test[wrong[i

                bayes_prob.append(prob)

            print 'The Bayes probability found is \n', bayes_prob
            print 'For this random example the model predicted a \n', misclassifie
            print ("Let's see how it compares to the actual image: {}").format(act
            plt.figure(1, figsize=(3, 3))
            plt.imshow(X_test[wrong[i]].reshape(28, 28), cmap=plt.cm.gray_r, inter
            plt.show()
```

```
The Bayes probability found is
[[0, -4086.2044056835412], [1, -4035.5495631430299], [2, -4071.9587276439979], [3,
For this random example the model predicted a
8
Let's see how it compares to the actual image: 5
```

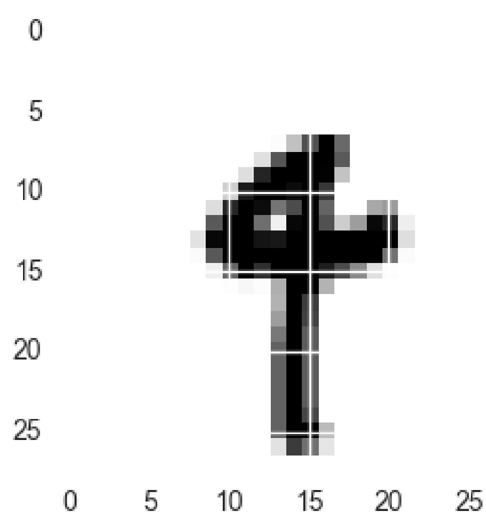



```
The Bayes probability found is
[[0, -4102.4500641906534], [1, -4151.553061514458], [2, -4033.8034182979213], [3, -
For this random example the model predicted a
```

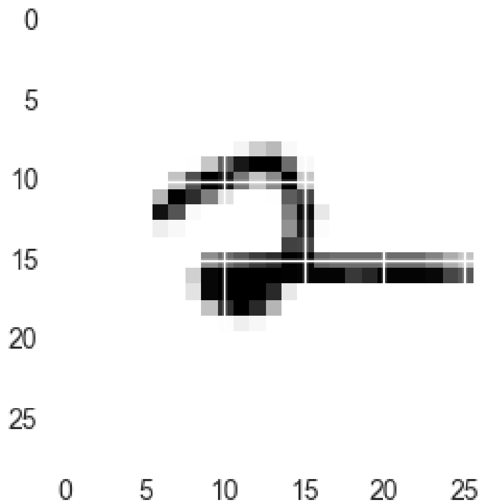
```
8
Let's see how it compares to the actual image: 2
```



```
The Bayes probability found is
[[0, -4174.8470824229125], [1, -4118.5923467660514], [2, -4117.8467729428721], [3,
For this random example the model predicted a
9
Let's see how it compares to the actual image: 4
```



```
The Bayes probability found is
[[0, -4132.2417508921853], [1, -4173.2826738035337], [2, -4049.3830047133774], [3,
For this random example the model predicted a
7
Let's see how it compares to the actual image: 2
```




```
The Bayes probability found is
[[0, -4064.0727425332047], [1, -4355.1813880862801], [2, -4068.8853861709608], [3,
For this random example the model predicted a
6
Let's see how it compares to the actual image: 4
```