DSE 210: final

Q4, Q6, Q7, Q8

In [2]:
```
%pylab inline
import numpy as np
import sklearn as sk
```

```
Populating the interactive namespace from numpy and matplotlib
```

Q4

For this problem, you'll be using the 20 Newsgroups data set. There are several versions of it on the web. You should download "20news-bydate.tar.gz" from http://qwone.com/~jason/20Newsgroups/ (http://qwone.com/~jason/20Newsgroups/) The same website has a processed version of the data, "20news-bydate-matlab.tgz", that is particularly convenient to use. Download this and also the file "vocabulary.txt". Look at the first training document in the processed set and the corresponding original text document to understand the relation between the two. The words in the documents constitute an overall vocabulary V of size 61188. Build a Bernoulli Naive Bayes model using the training data. Write a routine that uses this naive Bayes model to classify a new document. To avoid underflow, work with logs rather than multiplying together probabilities.

In [2]:
```
!curl -O http://qwone.com/~jason/20Newsgroups/20news-bydate-matlab.tgz
!tar xzvf 20news-bydate-matlab.tgz
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:02 --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:03 --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:04 --:--:--     0
100 7398k  100 7398k    0     0  1673k      0  0:00:04  0:00:04 --:--:-- 1678k
```

```
20news-bydate/matlab/
20news-bydate/matlab/train.data
20news-bydate/matlab/train.label
20news-bydate/matlab/train.map
20news-bydate/matlab/test.data
20news-bydate/matlab/test.label
20news-bydate/matlab/test.map
```

```
      0 [main] tar 14144 find_fast_cwd: WARNING: Couldn't compute FAST_CWD poin
ter.  Please report this problem to
the public mailing list cygwin@cygwin.com
```

In [3]:
```
data_train = np.loadtxt('20news-bydate/matlab/train.data',dtype='int')
Y_train = np.loadtxt('20news-bydate/matlab/train.label',dtype='int')
data_test = np.loadtxt('20news-bydate/matlab/test.data',dtype='int')
Y_test = np.loadtxt('20news-bydate/matlab/test.label',dtype='int')
max_doc,max_word,max_count = np.amax(data_train, axis=0)
max_word = 61188
X_train = np.zeros((max_doc,max_word), dtype=np.dtype('b'))
for d in data_train:
    doc,word = d[0],d[1]
    X_train[doc-1][word-1] = 1

max_doc_test,dummy1,dummy2 = np.amax(data_test, axis=0)
X_test = np.zeros((max_doc_test,max_word), dtype=np.dtype('b'))
for d in data_test:
    doc,word = d[0],d[1]
    X_test[doc-1][word-1] = 1

print X_train.shape
print X_test.shape

class_names = []
with open('20news-bydate/matlab/train.map') as f:
    class_names = [l.split(' ')[0] for l in iter(f)]
```

```
(11269L, 61188L)
(7505L, 61188L)
```

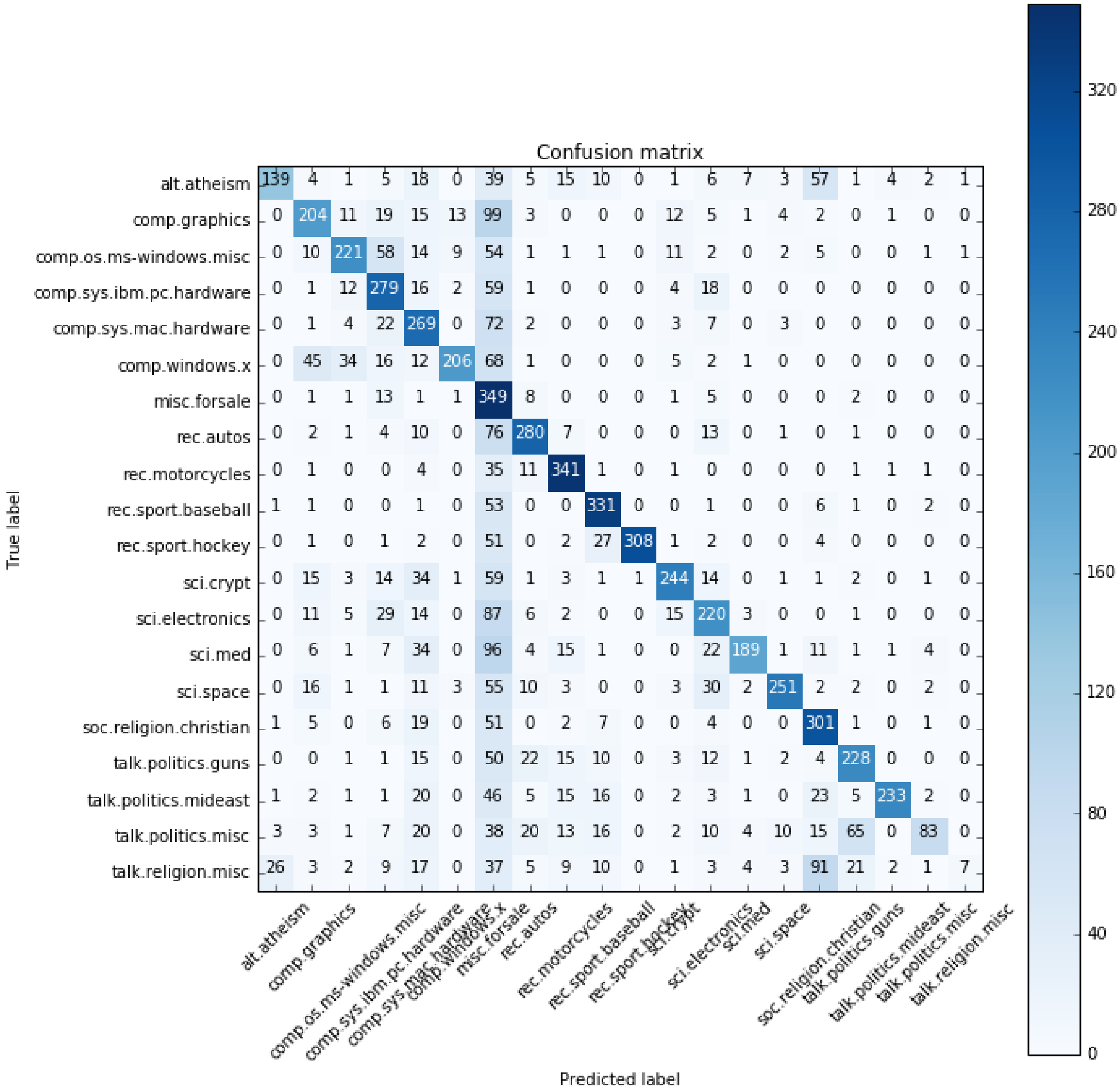
In [14]:
```
# (a) Evaluate the performance of your model on the test data. What error rate do
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
clf.fit(X_train, Y_train)
Y_predict = clf.predict(X_test)
errors = np.sum(Y_test != Y_predict)
print "error rate is:", float(errors) / len(Y_test)
```

```
error rate is: 0.37601598934
```

In [10]:
```
# (b) Evaluate your final model on the test data. Construct a confusion matrix.

import itertools
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
cnf_matrix = confusion_matrix(Y_test, Y_predict)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure(figsize=(10, 10))
plot_confusion_matrix(cnf_matrix, classes=class_names)
plt.show()
```

```
Confusion matrix, without normalization
[[139   4   1   5  18   0  39   5  15  10   0   1   6   7   3  57   1   4
    2   1]
 [  0 204  11  19  15  13  99   3   0   0   0  12   5   1   4   2   0   1
    0   0]
 [  0  10 221  58  14   9  54   1   1   1   0  11   2   0   2   5   0   0
    1   1]
 [  0   1  12 279  16   2  59   1   0   0   0   4  18   0   0   0   0   0
    0   0]
 [  0   1   4  22 269   0  72   2   0   0   0   3   7   0   3   0   0   0
```

```
   0   0]
 [  0  45  34  16  12 206  68   1   0   0   0   5   2   1   0   0   0   0
    0   0]
 [  0   1   1  13   1   1 349   8   0   0   0   1   5   0   0   0   2   0
    0   0]
 [  0   2   1   4  10   0  76 280   7   0   0   0  13   0   1   0   1   0
    0   0]
 [  0   1   0   0   4   0  35  11 341   1   0   1   0   0   0   0   1   1
    1   0]
 [  1   1   0   0   1   0  53   0   0 331   0   0   1   0   0   6   1   0
    2   0]
 [  0   1   0   1   2   0  51   0   2  27 308   1   2   0   0   4   0   0
    0   0]
 [  0  15   3  14  34   1  59   1   3   1   1 244  14   0   1   1   2   0
    1   0]
 [  0  11   5  29  14   0  87   6   2   0   0  15 220   3   0   0   1   0
    0   0]
 [  0   6   1   7  34   0  96   4  15   1   0   0  22 189   1  11   1   1
    4   0]
 [  0  16   1   1  11   3  55  10   3   0   0   3  30   2 251   2   2   0
    2   0]
 [  1   5   0   6  19   0  51   0   2   7   0   0   4   0   0 301   1   0
    1   0]
 [  0   0   1   1  15   0  50  22  15  10   0   3  12   1   2   4 228   0
    0   0]
 [  1   2   1   1  20   0  46   5  15  16   0   2   3   1   0  23   5 233
    2   0]
 [  3   3   1   7  20   0  38  20  13  16   0   2  10   4  10  15  65   0
   83   0]
 [ 26   3   2   9  17   0  37   5   9  10   0   1   3   4   3  91  21   2
    1   7]]
```

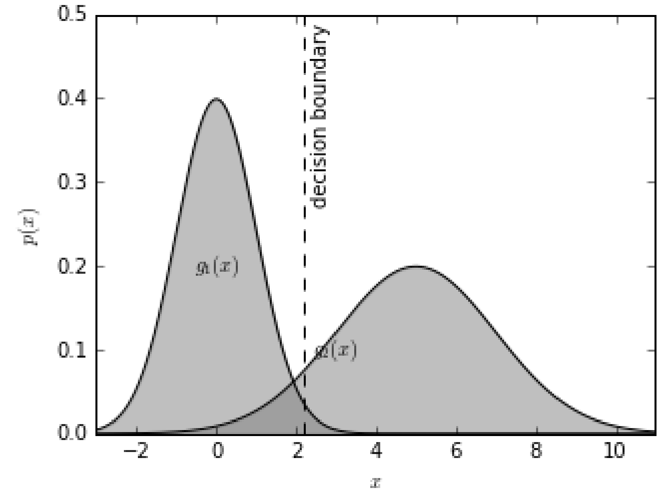


Q6

Urn A contains a Gaussian pdf: N(0, σ=1); Urn B contains another Gaussian N(5,σ=2); We draw a number and it is X=2.5. P(UrnA)=2P(UrnB) Determine the likelihood of (UrnA | X=2.5)? Determine a decision boundary (Urn A vs Urn B) for this problem.

In [16]:
```
import numpy as np
from scipy.stats import norm

p_X_A = norm(0, 1).pdf(2.5)
p_X_B = norm(5, 2).pdf(2.5)
posterior_AX = p_X_A * 2/3
posterior_BX = p_X_B * 1/3
likelihood_A_X = posterior_AX / (posterior_AX + posterior_BX)
print "likehood of (UrnA | X=2.5) is ", likelihood_A_X

# Compute the two PDFs
x = np.linspace(-3, 12, 1000)
pdf1 = norm(0, 1).pdf(x)
pdf2 = norm(5, 2).pdf(x)
x_bound = x[np.where(2*pdf1 < pdf2)][0]

# Plot the pdfs and decision boundary
fig = plt.figure(figsize=(5, 3.75))
ax = fig.add_subplot(111)
ax.plot(x, pdf1, '-k', lw=1)
ax.fill_between(x, pdf1, color='gray', alpha=0.5)

ax.plot(x, pdf2, '-k', lw=1)
ax.fill_between(x, pdf2, color='gray', alpha=0.5)

# plot decision boundary
ax.plot([x_bound, x_bound], [0, 0.5], '--k')

ax.text(x_bound + 0.2, 0.49, "decision boundary",
        ha='left', va='top', rotation=90)

ax.text(0, 0.2, '$g_1(x)$', ha='center', va='center')
ax.text(3, 0.1, '$g_2(x)$', ha='center', va='center')

ax.set_xlim(-3, 11)
ax.set_ylim(0, 0.5)

ax.set_xlabel('$x$')
ax.set_ylabel('$p(x)$')

plt.show()
print "decision boundary is: ", x_bound
```

```
likehood of (UrnA | X=2.5) is  0.277387907451
```




```
decision boundary is:  2.1951951952
```

Q7

Consider the following observations: X=(-0.1,-0.2, 0.1, 0.2, 0, 0.1, -0.1, 0, -0.05, 0.1, 1.05, 1.1, 0.9, 0.8, 0.9, 1, 1.2, 1.1,1.2, .9) Cluster this data into two classes using the K-means algorithm. What are the cluster centers?

In [5]:
```
from sklearn.cluster import KMeans
X=np.array([-0.1,-0.2, 0.1, 0.2, 0, 0.1, -0.1, 0, -0.05, 0.1, 1.05, 1.1, 0.9, 0.8
X = X.reshape(-1, 1)
estimator = KMeans(init='k-means++', n_clusters=2)
estimator.fit(X.reshape(-1, 1))

print "labels for the test data:\n", estimator.labels_
print "cluster centers:\n", estimator.cluster_centers_
```

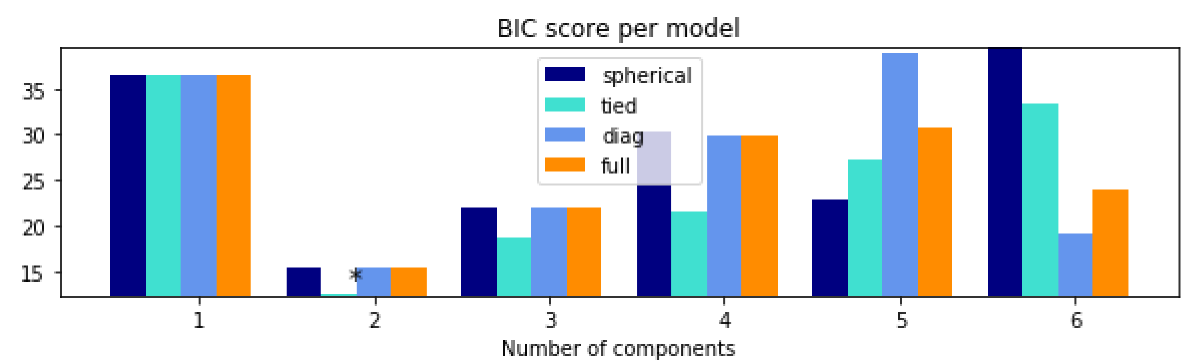
```
labels for the test data:
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
cluster centers:
[[ 0.005]
 [ 1.015]]
```

Q8

Construct a Gaussian Mixture Model for the above data. What is your estimate for the number of mixtures?

In [14]:
```
import itertools

from sklearn import mixture

# Number of samples per component
lowest_bic = np.infty
bic = []
n_components_range = range(1, 7)
cv_types = ['spherical', 'tied', 'diag', 'full']
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit a Gaussian mixture with EM
        gmm = mixture.GaussianMixture(n_components=n_components,
                                      covariance_type=cv_type)
        gmm.fit(X)
        bic.append(gmm.bic(X))
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

bic = np.array(bic)
color_iter = itertools.cycle(['navy', 'turquoise', 'cornflowerblue',
                              'darkorange'])
clf = best_gmm
bars = []

# Plot the BIC scores
plt.figure(figsize=(10, 5))
spl = plt.subplot(2, 1, 1)
for i, (cv_type, color) in enumerate(zip(cv_types, color_iter)):
    xpos = np.array(n_components_range) + .2 * (i - 2)
    bars.append(plt.bar(xpos, bic[i * len(n_components_range):
                                  (i + 1) * len(n_components_range)],
                        width=.2, color=color))
plt.xticks(n_components_range)
plt.ylim([bic.min() * 1.01 - .01 * bic.max(), bic.max()])
plt.title('BIC score per model')
xpos = np.mod(bic.argmin(), len(n_components_range)) + .65 +\
    .2 * np.floor(bic.argmin() / len(n_components_range))
plt.text(xpos, bic.min() * 0.97 + .03 * bic.max(), '*', fontsize=14)
spl.set_xlabel('Number of components')
spl.legend([b[0] for b in bars], cv_types)
```

Out[14]: <matplotlib.legend.Legend at 0xc958208>



Based on BIC score above, 2 mixtures seem to be good estimate