hw3

March 2, 2015

1 Homework 1

- Name: Austin Chen
- SID: 23826762
- Repro: Open up hw1.ipynb in IPython Notebook.

1.1 Problem 1

```
In [141]: import numpy as np
          samples1 = np.random.normal(3, np.sqrt(9), 100)
          samples2 = np.random.normal(3, np.sqrt(9), 100) / 2 + np.random.normal(4, np.sqrt(4), 100)
```

1.1.1 (a) Averages

```
In [142]: u1 = np.average(samples1)
          u1

Out[142]: 2.3992975413386386

In [143]: u2 = np.average(samples2)
          u2

Out[143]: 5.3880865163180331
```

1.1.2 (b) Covariance matrix

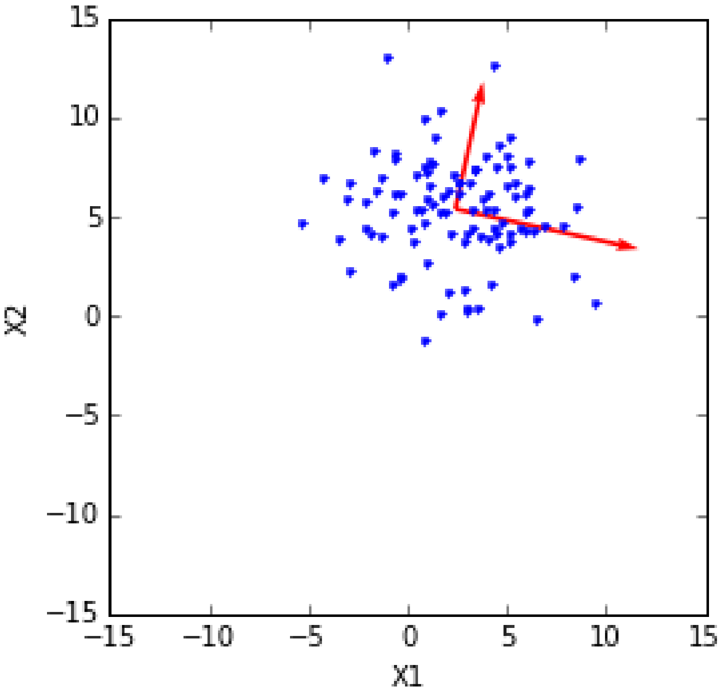
```
In [144]: cov_matrix = np.cov(samples1, samples2)
          cov_matrix

Out[144]: array([[ 9.33321311, -0.60475554],
                 [-0.60475554,  6.69091187]])
```

1.1.3 (c) Eigenvectors and eigenvalues

```
In [145]: eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
          eigenvalues

Out[145]: array([ 9.46504846,  6.55907652])

In [146]: eigenvectors

Out[146]: array([[ 0.9770532,  0.2129954],
                 [-0.2129954,  0.9770532]])
```

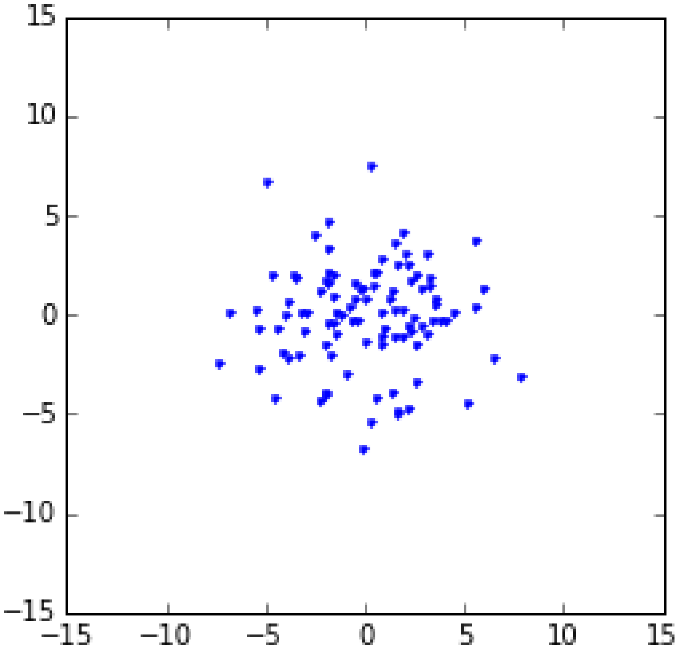
1.1.4 (d) Plot

```
In [147]: %matplotlib inline
          import matplotlib.pyplot as plt

          # Show the samples
          plt.plot(samples1, samples2, '.')
          plt.axis([-15, 15, -15, 15])
          plt.axes().set_aspect('equal')
          plt.xlabel('X1')
          plt.ylabel('X2')

          # Show the covariance eigenvectors
          U, V = zip(eigenvectors[:,0] * eigenvalues[0], eigenvectors[:,1] * eigenvalues[1])
          plt.quiver([u1, u1], [u2, u2], U, V, color='r', angles='xy', scale_units='xy', scale=1)

          plt.show()
```



1.1.5 (e) Plot, rotated

```
In [148]: # Rotate the points
          inverted = np.vstack((eigenvectors[:,1], eigenvectors[:,0])).T
          U = eigenvectors if eigenvalues[0] > eigenvalues[1] else inverted
          r1, r2 = np.dot(U.T, np.vstack((samples1 - u1, samples2 - u2)))

          # Then plot them
          plt.plot(r1, r2, '.')
          plt.axis([-15, 15, -15, 15])
          plt.axes().set_aspect('equal')
          plt.show()
```



1.2 Problem 2

1.2.1 (a)

If one of the components of the random vector X is a linear combination of the others, then the covariance matrix will not be full rank and thus will be singular. We can drop that component from X, reducing its dimensionality and restoring the covariance matrix to full rank and thus invertibility.

1.2.2 (b)

$$x^T\Sigma^{-1}x$$

$$= x^T U \Lambda^{-1} U^T x$$

$$= \sum_i \frac{(v_i^T * x)^2}{\lambda_i}$$

Thus, A is the matrix where the ith row is $v_i/\sqrt{\lambda_i}$.

1.2.3 (c)

Multiplication by A is analagous to multiplying by an orthonormal matrix followed by a diagonal one, which is analogous to a rotation then a scaling of each component.

$||Ax||_2^2$ controls the likelihood of x appearing in a particular Gaussian. That is to say, if the vector that is Ax is long, then x is unlikely to appear; if Ax is short, then x is more likely to appear.

1.2.4 (d)

If the components of the random vector X are independent, then A is diagonal, and the minimum value of $||Ax||_2^2$ is when the largest scalings done by A are applied on the smallest components of x and vice versa. Conversely, the maximum is achieved when the largest scalings are applied on the largest components of x.

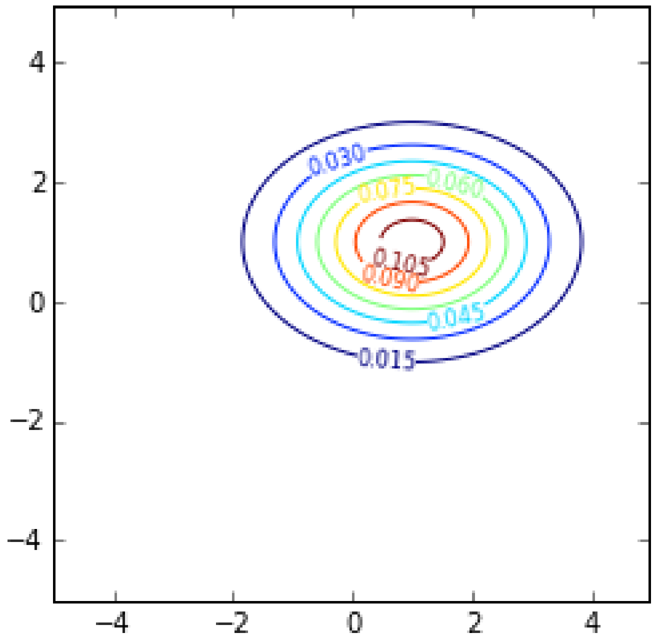
To maximize f(x), we should choose the x that minimizes $||Ax||_2^2$.

1.3 Problem 3

```
In [212]: # http://matplotlib.org/examples/pylab_examples/contour_demo.html
          import matplotlib.mlab as mlab

          delta = 0.025
          x = np.arange(-5.0, 5.0, delta)
          y = np.arange(-5.0, 5.0, delta)
          X, Y = np.meshgrid(x, y)

          def isocontour(Z):
              CS = plt.contour(X, Y, Z)
              plt.clabel(CS, inline=1, fontsize=8)
              plt.axes().set_aspect('equal')
```

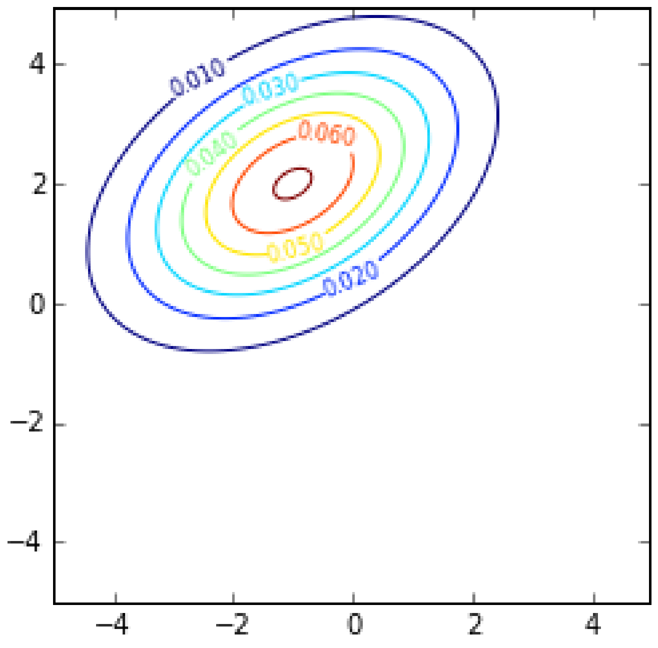
1.3.1 (a)

```
In [205]: Z = mlab.bivariate_normal(X, Y, sigmax=np.sqrt(2), sigmay=1, mux=1, muy=1, sigmaxy=0)
          isocontour(Z)
```



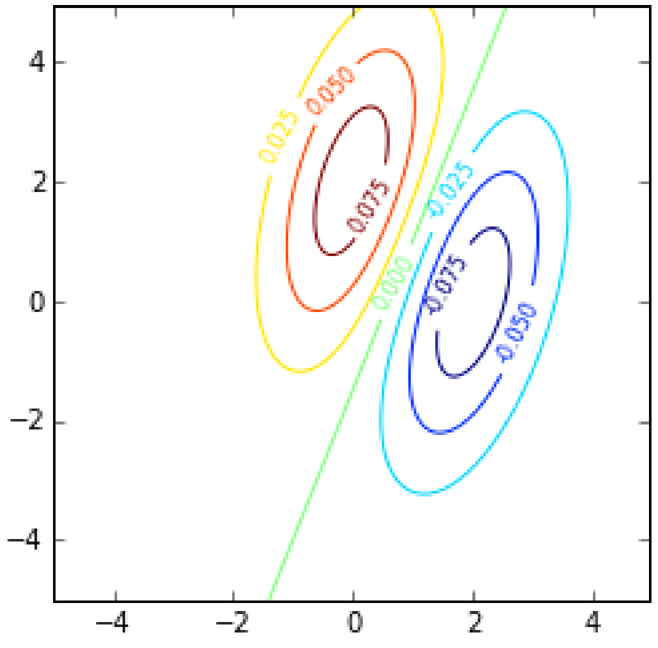
1.3.2 (b)

```
In [206]: Z = mlab.bivariate_normal(X, Y, sigmax=np.sqrt(3), sigmay=np.sqrt(2), mux=-1, muy=2, sigmaxy=
          isocontour(Z)
```



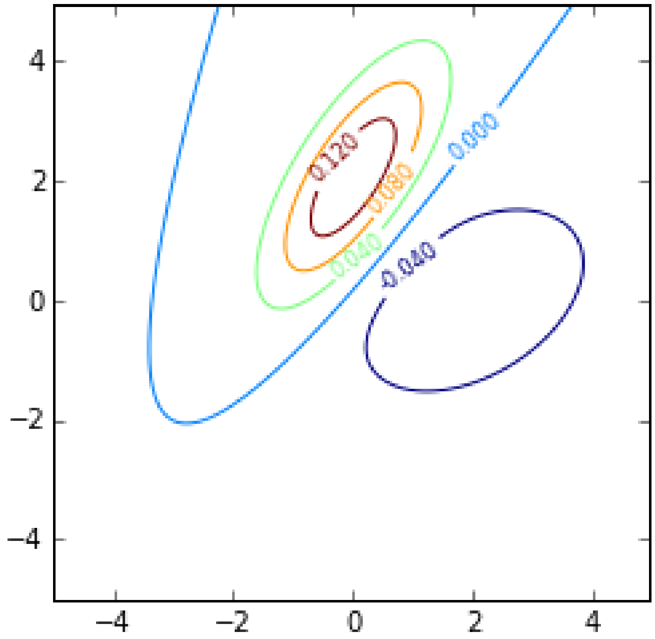
1.3.3 (c)

```
In [152]: Z1 = mlab.bivariate_normal(X, Y, sigmax=1, sigmay=2, mux=0, muy=2, sigmaxy=1)
          Z2 = mlab.bivariate_normal(X, Y, sigmax=1, sigmay=2, mux=2, muy=0, sigmaxy=1)
          isocontour(Z1 - Z2)
```



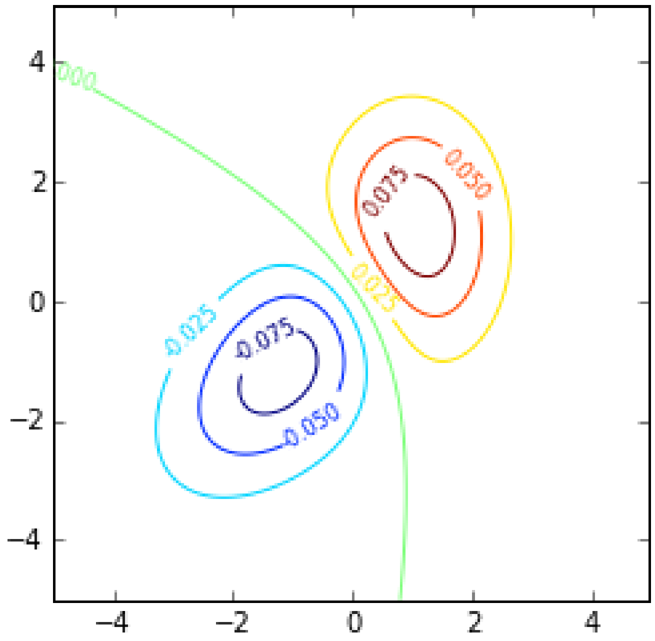
1.3.4 (d)

```
In [208]: Z1 = mlab.bivariate_normal(X, Y, sigmax=1, sigmay=np.sqrt(2), mux=0, muy=2, sigmaxy=1)
          Z2 = mlab.bivariate_normal(X, Y, sigmax=np.sqrt(3), sigmay=np.sqrt(2), mux=2, muy=0, sigmaxy=
          isocontour(Z1 - Z2)
```



1.3.5 (e)

```
In [213]: Z1 = mlab.bivariate_normal(X, Y, sigmax=1, sigmay=np.sqrt(2), mux=1, muy=1, sigmaxy=0)
          Z2 = mlab.bivariate_normal(X, Y, sigmax=np.sqrt(2), sigmay=np.sqrt(2), mux=-1, muy=-1, sigmax
          isocontour(Z1 - Z2)
```



1.4 Problem 4

```
In [201]: # Load the data
          import scipy.io
          mat = scipy.io.loadmat('data/digit-dataset/train.mat')
          images = np.reshape(mat['train_image'], (1, -1, 60000))[0].T
          labels = mat['train_label']

          # Shuffle the data in parallel
          p = np.random.permutation(len(images))
          images, labels = images[p], labels[p]
```

```
In [191]: from scipy.stats import multivariate_normal as m
          from sklearn.preprocessing import normalize

          def make_classifier(images, labels, average_cov):
              # Model each class as a Gaussian
              classes = list(set(labels[:,0]))
              means, covs, priors = [], [], []
              for c in classes:
                  # Filter out the class, then normalize
                  locs = np.array([label[0] == c for label in labels])
                  data = normalize(images[locs].astype(float), norm='l2')

                  # Calculate mean, covariance, and priors
                  means.append(sum(data) / len(data))
                  covs.append(np.cov(data, rowvar=0))
                  priors.append(len(data) / len(images))

              # Class conditional distributions i.e. P(x|c)
```

```
    if average_cov:
        overall = sum(covs) / len(covs)
        rvs = [m(mean, overall, True) for mean in means]
    else:
        rvs = [m(mean, cov, True) for mean, cov in zip(means, covs)]

    # Posterior probability i.e. P(x|c)*P(c)
    def posterior(image, c):
        return rvs[c].logpdf(image) + np.log(priors[c])

    # Return the trained classifying function
    def classify(image):
        return max(classes, key=lambda c: posterior(image, c))
    return classify

def score(images, labels, tests, answers, average_cov):
    classify = make_classifier(images, labels, average_cov)
    correct = sum(classify(t) == a for t, a in zip(tests, answers))
    return correct / len(tests)
```

1.4.1 (a)

The maximum likelihood estimator of a multivariate Gaussian's mean is given by:

$$l(\mu, \Sigma) = -\frac{nd}{2} log(2\pi) - \frac{n}{2} log|\Sigma| - \frac{1}{2} \sum\nolimits_i (x_i - \mu)^T \Sigma^{-1} (x_i - \mu)$$

$\nabla l(\mu) = 0$ at:

$$\mu = \frac{1}{n} \sum\nolimits_i x_i$$

$\nabla l(\Sigma) = 0$ at:

$$\Sigma = \frac{1}{n} \sum\nolimits_i (x_i - \mu)(x_i - \mu)^T$$

This estimator for the mean is unbiased, since its expectation is the same as the actual mean. Covariance is not.

1.4.2 (b)

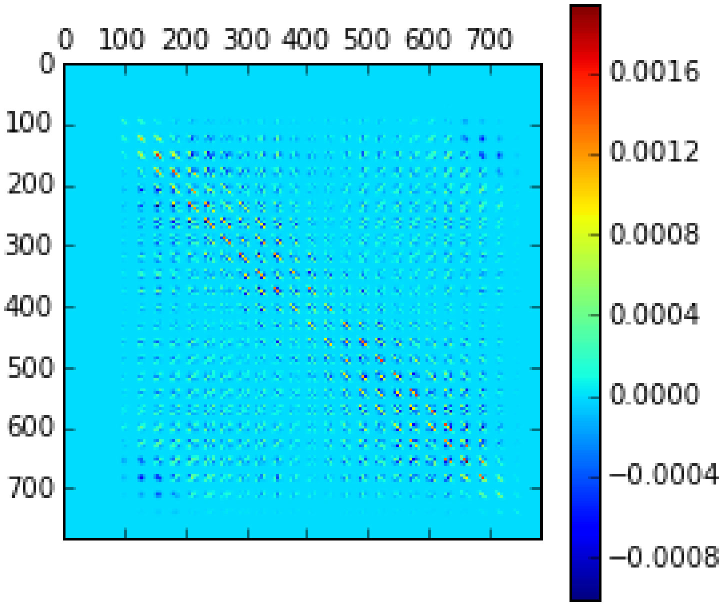
The priors for each class can be approximated by the frequency of that class in the training labels. For example, if the 11% of the training labels were '8', we could say that the prior probability of an '8' is 0.11.

1.4.3 (c)

The covariance matrix is mostly 0, indicating most pixels are independent of most others. However, there are nonzero covariances in a band around the diagonal, meaning that pixels are not indepedent of pixels in their near vicinity. Also, there's a spot in the top right and bottom left – possibly because the top and bottom of 8s are correlated.

```
In [192]: # Generate covariance matrix for 8s
          locs = np.array([label[0] == 8 for label in labels])
          data = normalize(images[locs].astype(float), norm='l2')
          cov = np.cov(data, rowvar=0)

          # Visualize
          plt.matshow(cov)
          plt.colorbar()
```

```
Out[192]: <matplotlib.colorbar.Colorbar at 0x10d4a130>
```

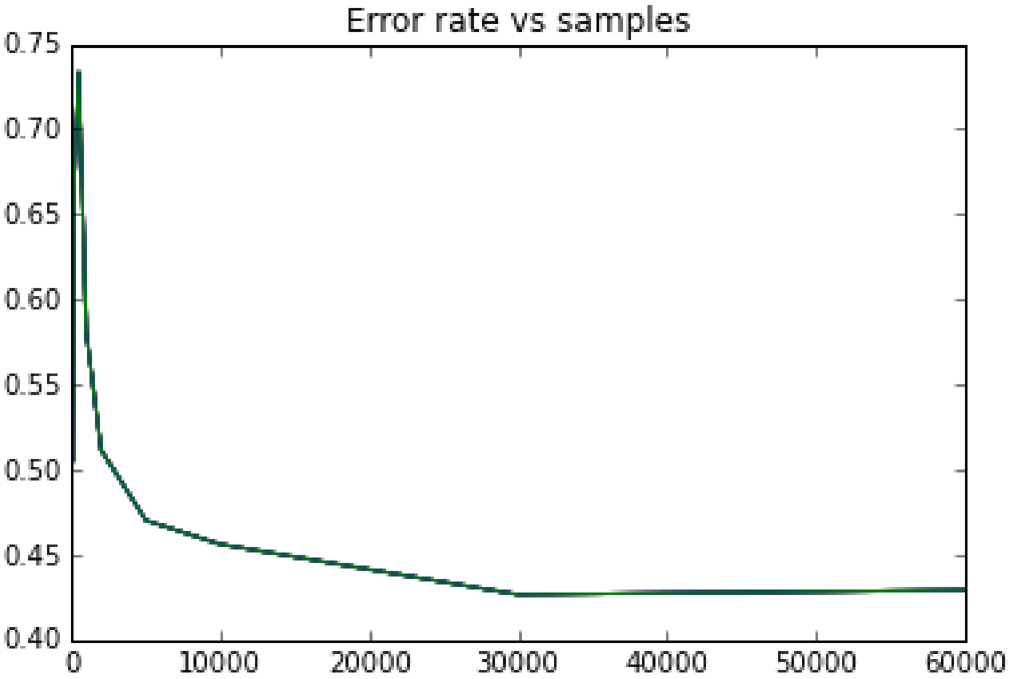


1.4.4 (d.i) Averaged covariance matrix

```
In [ ]: # Load the test data
        mat = scipy.io.loadmat('data/digit-dataset/test.mat')
        test = np.reshape(mat['test_image'], (1, -1, 5000))[0].T
        answers = mat['test_label']
```

```
In [ ]: x = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 60000]
        y1 = [score(images[:size], labels[:size], test, answers, True) for size in x]
```

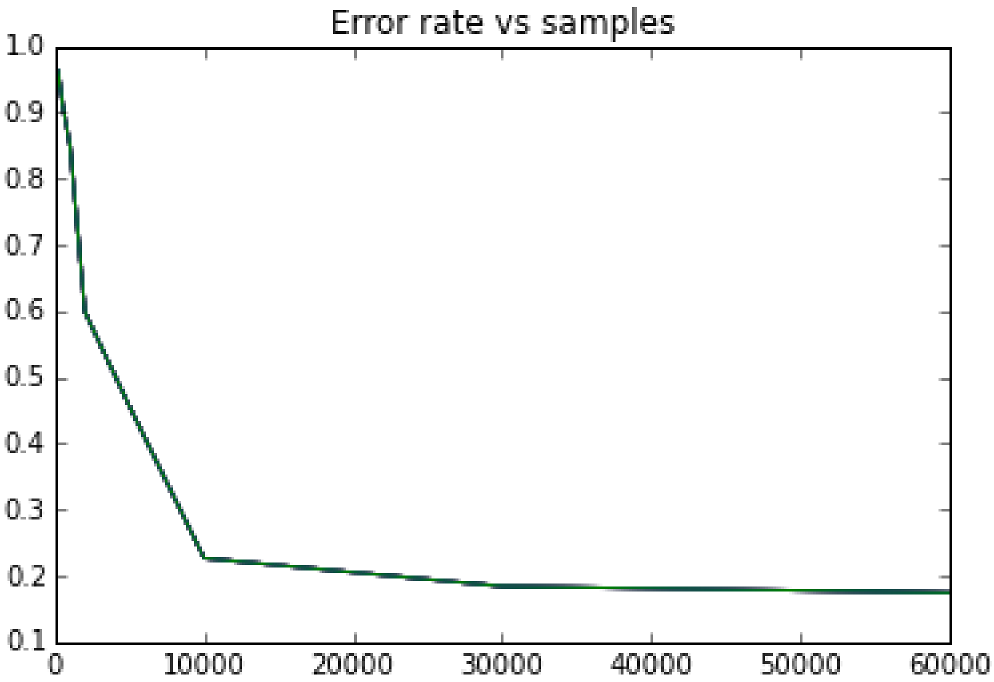
```
In [160]: plt.plot(x, np.ones(len(y1)) - y1)
          plt.title('Error rate vs samples')
          plt.show()
```



1.4.5 (d.ii) Unique covariance matrices

```
In [161]: x = [100, 200, 500, 1000, 2000, 5000, 10000, 30000, 60000]
          y2 = [score(images[:size], labels[:size], test, answers, False) for size in x]
```

```
In [162]: plt.plot(x, np.ones(len(y2)) - y2)
          plt.title('Error rate vs samples')
          plt.show()
```



1.4.6 (d.iv) Kaggle for digits

My Kaggle score was 0.82120.

```
In [163]: # Load the kaggle data
          mat = scipy.io.loadmat('data/digit-dataset/kaggle.mat')
          kaggle = np.reshape(mat['kaggle_image'], (1, -1, 5000))[0].T

          # Run the classifier trained on all the training data
          classify = make_classifier(images, labels, False)
          result = [classify(test) for test in kaggle]

          # Write the results to a csv
          f = open('digits.csv', 'w')
          f.write('Id,Category\n')
          for i in range(len(result)):
              f.write("{0},{1}\n".format(i + 1, result[i]))
          f.close()
```

1.4.7 (e) Kaggle for spam

My Kaggle score was 0.76844.

```
In [164]: # Load the data
          mat = scipy.io.loadmat('data/spam-dataset/spam_data.mat')
          emails = mat['training_data']
          labels = (mat['training_labels']).T
          kaggle = mat['test_data']
```

```
# Run the classifier trained on all the training data
classify = make_classifier(emails, labels, False)
result = [classify(test) for test in kaggle]

# Write the results to a csv
f = open('spam.csv', 'w')
f.write('Id,Category\n')
for i in range(len(result)):
    f.write("{0},{1}\n".format(i + 1, result[i]))
f.close()
```

1.5 Problem 5

To optimize the loss function J, set the gradient to 0 and solve for $\vec{w}$ and w_0.

$$J(\vec{w}, w_0) = (\vec{y} - \vec{X}\vec{w} - w_0\vec{1})^T(\vec{y} - \vec{X}\vec{w} - w_0\vec{1}) + \lambda\vec{w}^T\vec{w}$$

Solving for w_0:

$$J(\vec{w}, w_0) = -w_0\vec{1}^T(\vec{y} - \vec{X}\vec{w}) - w_0(\vec{y}^T - (\vec{X}\vec{w})^T)\vec{1} + w_0^2\vec{1}^T\vec{1} + ...$$

$$\nabla J(w_0) = -\vec{1}^T(\vec{y} - \vec{X}\vec{w}) - (\vec{y}^T - (\vec{X}\vec{w})^T)\vec{1} + 2w_0\vec{1}^T\vec{1} = 0$$

$$2w_0\vec{1}^T\vec{1} = \vec{1}^T\vec{y} + \vec{1}^T\vec{X}\vec{w} + \vec{y}^T\vec{1} + \vec{w}^T\vec{X}^T\vec{1}$$

Note that $\vec{1}^T\vec{y} = \vec{y}^T\vec{1} = \sum_i y_i$, and $\vec{1}^T\vec{X} = \vec{X}^T\vec{1} = \sum_i \vec{x_i} = 0$.

$$nw_0 = \sum_i y_i$$

$$w_0 = \bar{y}$$

Solving for $\vec{w}$:

$$J(\vec{w}, w_0) = -\vec{y}^T\vec{X}\vec{w} + \vec{w}^T\vec{x}^T\vec{X}\vec{w} + w_0 1^T\vec{X}\vec{w} - \vec{w}^T\vec{X}^T\vec{y} + w_0\vec{w}^T\vec{X}^T 1 + \lambda\vec{w}^T\vec{w} + ...$$

$$\nabla J(\vec{w}) = (\vec{X}^T\vec{X} + (\vec{X}^T\vec{X})^T)\vec{w} - 2\vec{y}^T\vec{X} + \lambda(I + I^T)\vec{w} + 2w_0\vec{X}^T 1 = 0$$

$$2\vec{X}^T\vec{X}\vec{w} + 2\lambda\vec{w} = 2\vec{y}^T\vec{X}$$

$$(\vec{X}^T\vec{X} + \lambda)\vec{w} = \vec{y}^T\vec{X}$$

$$\vec{w} = (\vec{X}^T\vec{X} + \lambda)^{-1}\vec{y}^T\vec{X}$$

1.6 Problem 6

To get the maximum likelihood, set the gradient to 0 and solve for w_0 and w_1. In log likelihood:

$$l(w_0, w_1) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (y_i - w_0 - w_1x_i)^2$$

Solving for w_0:

$$\nabla l(w_0) = \frac{1}{2\sigma^2}\sum_i 2(y_i - w_0 - w_1x_i) = 0$$

$$\sum_i y_i - w_0 - w_1x_i = 0$$

$$\sum_i w_0 = \sum_i y_i - w_1x_i$$

$$nw_o = \sum_i y_i - w_1x_i$$

$$w_0 = \frac{1}{n}\sum\nolimits_i y_i - w_1\frac{1}{n}\sum\nolimits_i x_i$$
$$w_0 = \bar{y} - w_1\bar{x}$$

Solving for w_1: Substitute in w_0 to solve for w_1:

$$l(w_0, w_1) = -\frac{n}{2}log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum\nolimits_i (y_i - \bar{y} + w_1\bar{x} - w_1 x_i)^2$$

$$\nabla l(w_1) = \frac{1}{2\sigma^2}\sum\nolimits_i 2(y_i - \bar{y} + w_1(\bar{x} - x_i))(\bar{x} - x_i) = 0$$

$$\sum\nolimits_i (y_i - \bar{y} + w_1(\bar{x} - x_i))(\bar{x} - x_i) = 0$$

$$\sum\nolimits_i w_1(x_i - \bar{x})^2 = \sum\nolimits_i (y_i - \bar{y})(x_i - \bar{x})$$

$$w_1\sum\nolimits_i (x_i - \bar{x})^2 = \sum\nolimits_i (y_i - \bar{y})(x_i - \bar{x})$$

$$w_1 = \frac{\sum_i (y_i - \bar{y})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

1.7 Problem 7

1.7.1 (a)

$$P(X = 0) = 1/2, \text{ and } P(Y = 1|X = 0) = P(Y = -1|X = 0) = 1/2.$$
$$P(Y = 0) = 1/2, \text{ and } P(X = 1|Y = 0) = P(X = -1|Y = 0) = 1/2.$$

Thus, the values that (X, Y) can take on are given as the 4 pairs (0, 1), (0, -1), (1, 0), (-1, 0), each with probability 1/4.

X and Y are uncorrelated. Proof: Consider $E[XY]$. XY is a random variable that is always 0, since there is a 0 component in every (x, y) pair.

Consider $E[X]$ and $E[Y]$. X takes on 0 with probability 1/2, 1 with probability 1/4, and -1 with probability 1/4, for a total expectation of 0. Y has the same distribution and thus expectation of 0.

$E[XY] - E[X]E[Y] = 0 - 0 = 0$, so X and Y are uncorrelated by definition.

X and Y are not independent. Proof: $P(X = x, Y = y) = P(X = x) * P(Y = y)$ must be true for all x, y, for X and Y to be independent. Consider:

$$P(X = 0, Y = 0) \overset{?}{=} P(X = 0) * P(Y = 0)$$

$$0 \neq 1/2 * 1/2$$

Thus X and Y are not independent.

1.7.2 (b)

Consider the following truth table:

B1	B2	B3	X	Y	Z
1	1	1	0	0	0
1	0	1	1	1	0
1	1	0	0	1	1
1	0	0	1	0	1
0	1	1	1	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	0	0	0	0	0

$P(X = x, Y = y) = P(X = x) * P(Y = y)$ must be true for all x, y. This is true by inspection of the truth table – every (x, y) pair occurs with probability 1/4, and P(x) and P(y) are 1/2 for every x and y. Thus, X and Y are independent. This is true by symmetry for X and Z, and Y and Z. Thus, X, Y, and Z are pairwise independent.

However, $P(x = 1, y = 1, z = 1) = 0$, whereas $P(x = 1) * P(y = 1) * P(z = 1) = 1/8$. Thus X, Y, and Z are not mutually independent.