CS189/289A – Spring 2017 — Homework 3

Yicheng Chen, SID 26943685

1. Independence v.s. Correlation

(a) From the description of the problem, we know that (X, Y) can be the following 4 states with equal probability:

$$(0, 1), (0, -1), (1, 0), (-1, 0)$$

Therefore,

$$Cov(X, Y) = \mathbb{E}(XY) - \mathbb{E}(X)\mathbb{E}(Y) = 0 - 0 = 0$$

So X and Y are uncorrelated.
However, let's examine $P(X = 0, Y = 1)$:

$$P(X = 0, Y = 1) = \frac{1}{4}$$

However,

$$P(X = 0)P(Y = 1) = \frac{1}{2} \times \frac{1}{4} = \frac{1}{8} \neq P(X = 0, Y = 1)$$

So X, Y are not indepenedent.

(b) It is obvious that

$$P(X = 1) = P(B = 1, C = 0) + P(B = 0, C = 1) = \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

Similarly, $P(X = 0) = P(Y = 1) = P(Y = 0) = P(Z = 1) = P(Z = 0) = \frac{1}{2}$.
And to test whether X, Y and Z are pairwise independent, for example,

$$P(X = 1, Y = 1) = \frac{1}{4} = P(X = 1)P(Y = 1)$$

Due to the symmetry of XOR, it is obvious that X, Y, Z are pairwise independent. However,

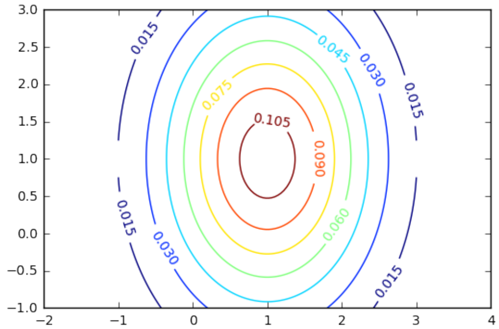
$$P(X = 1, Y = 1, Z = 1) = 0$$

Because no combination of B, C, D can make all of X, Y, Z to be 1. But

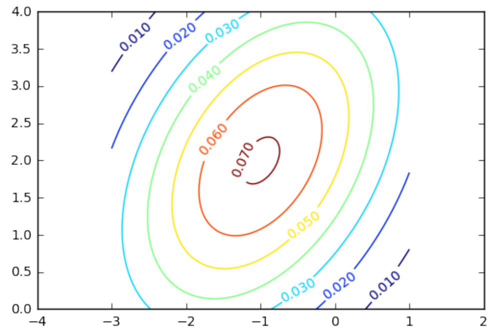
$$P(X = 1)P(Y = 1)P(Z = 1) = \frac{1}{8} \neq P(X = 1, Y = 1, Z = 1)$$

So they are not mutually independent.

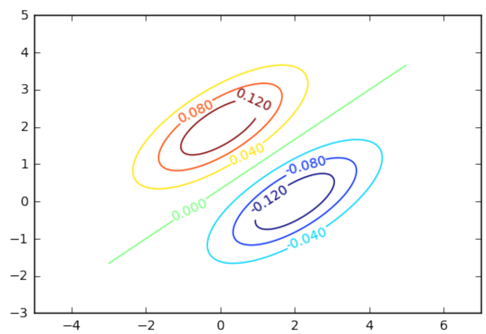
2. Isocontours of Normal Distributions



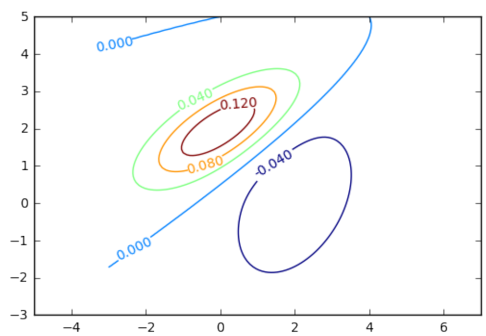
(a)



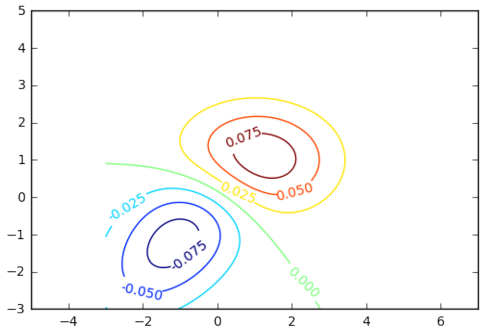
(b)



(c)



(d)

(e) 

3. Eigenvectors of the Gaussian Covariance Matrix

(a) Mean of the samples:

$$(3.55073982, 5.84907657)$$

(b) Covariance matrix of the samples:

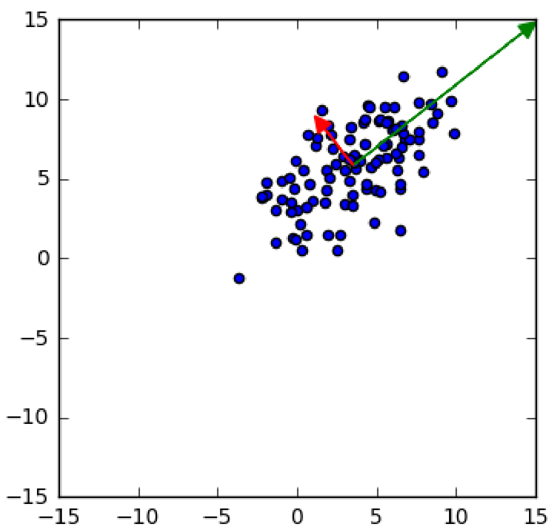
$$\begin{bmatrix} 9.53074878 & 5.16187852 \\ 5.16187852 & 6.99366966 \end{bmatrix}$$

(c) Eigenvectors and eigenvalues of the covariance matrix:

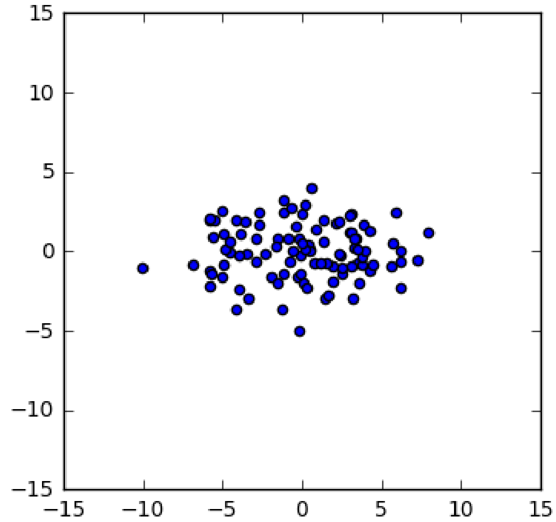
$$\lambda_1 = 13.57767557, v_1 = \begin{bmatrix} 0.78697226 \\ 0.61698839 \end{bmatrix}$$

$$\lambda_2 = 2.94674287, v_2 = \begin{bmatrix} -0.61698839 \\ 0.78697226 \end{bmatrix}$$

(d) Scatter plot and arrows representing eigenvectors/eigenvalues:



(e) Rotated Points:


15
10
5
0
−5
−10
−15
−15 −10 −5 0 5 10 15

4. Maximum Likelihood Estimation

(a) The log likelihood function:

$$l(\mu, \Sigma; X_1, X_2, ...X_n) = \sum_{i=1}^{n}(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log|\Sigma|)$$

Taking the derivative with respect to μ and setting it to zero:

$$\frac{\partial l}{\partial \mu} = \sum_{i=1}^{n}(X_i-\mu)^\top \Sigma^{-1} = 0$$

So

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Taking the derivative with respect to Σ^{-1} and setting it to zero:

$$\frac{\partial l}{\partial \Sigma^{-1}} = \frac{\partial}{\partial \Sigma^{-1}} \frac{n}{2}\log|\Sigma^{-1}| - \frac{1}{2}\sum_{i=1}^{n}\mathrm{tr}[(X_i-\mu)(X_i-\mu)^\top \Sigma^{-1}]$$

$$= \frac{n}{2}\Sigma - \frac{1}{2}\sum_{i=1}^{n}(X_i-\mu)(X_i-\mu)^\top = 0$$

So

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(X_i-\hat{\mu})(X_i-\hat{\mu})^\top$$

Therefore,

$$\hat{\sigma_j} = \sqrt{\sum_{i=1}^{n}(X_{ij}-\mu_j)^2}$$

(b) The log likelihood function:

$$l(\mu, \Sigma; X_1, X_2, ...X_n) = \sum_{i=1}^{n}(-\frac{1}{2}(x-A\mu)^\top \Sigma^{-1}(x-A\mu) - \frac{d}{2}\log 2\pi - \frac{1}{2}\log|\Sigma|)$$

Taking the derivative with respect to μ and setting it to zero:

$$\frac{\partial l}{\partial \mu} = A^\top \sum_{i=1}^{n}(X_i-A\mu)^\top \Sigma^{-1} = 0$$

So,

$$\hat{\mu} = \frac{A^{-1}\sum_{i=1}^{n} X_i}{n}$$

5. Covariance Matrices and Decompositions

(a) When $n < d$ or there are lees than d linearly independent vector $(X_i - \mu)$, $\hat{\Sigma}$ would be singular.

(b) We can get a nonsingular covariance matrix estimator from the original one by adding a small multiple of identity matrix:

$$\hat{\Sigma}' = \hat{\Sigma} + xI$$

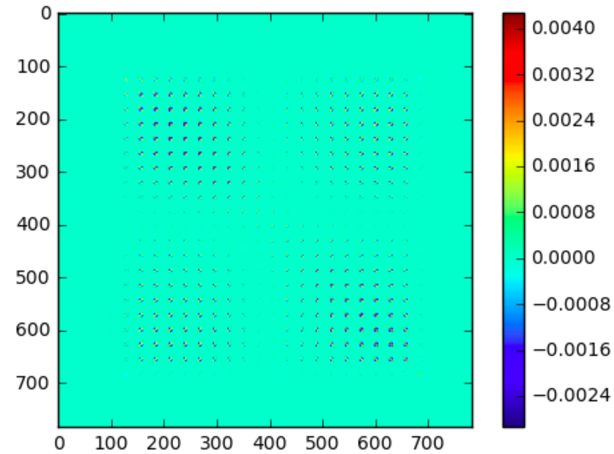
To avoid changing the covariance matrix too much, the parameter x can be set to one or two order of magnitude smaller than the minimum non-zero eigenvalue of the original covariance matrix estimator. In this way, the matrix eigenvalues are not changed too much.

(c) The vector that is the eigenvector with respect to the smallest eigenvalue of Σ would maximize the PDF $f(x)$. Similarly, the vector that is the eigenvector with respect to the largest eigenvalue of Σ would minimize $f(x)$.

6. Gaussian Classifiers for Digits and Spam

(a) See code appendix.

(b) Cov matrix for digit 1:



The diagonal terms seem to be larger than non-diagonal terms.

(c) LDA and QDA see code appendix.
QDA performs slightly better because the boundary is more flexible.
MNISt–Kaggle score: 0.95440 (Name: Yicheng Chen)

(d) Spam–Kaggle score: 0.76780 (Name: Yicheng Chen)

Appendix

1. Code for Problem 1 to 5:

```
# coding: utf-8

# ## CS189/289A HW3 code part 1
# ### Yicheng Chen
# ### 02/15/2017
#

# ---
# ### Q2

# In[9]:

import numpy as np
import matplotlib.pyplot as plt
import scipy.interpolate
import math


# In[33]:

def gaussian_2d(miu, cov, x, y):
    pi=3.1415926
    pos = np.array([x, y])
    S = np.linalg.det(cov)
    C = np.linalg.inv(cov)
    g = 1/(2*pi*math.sqrt(S)) * math.exp(-0.5*(pos-miu).dot(C.dot(pos-miu)))
    return g


# In[45]:

# a
xi, yi = np.linspace(-1, 3, 100), np.linspace(-1, 3, 100)
xi, yi = np.meshgrid(xi, yi)
miu = np.array([1, 1])
cov = np.array([[1, 0], [0, 2]])
z = np.array([gaussian_2d(miu, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).re
CS = plt.contour(z, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(CS, inline=1, fontsize=10)
plt.axis('equal')
plt.show()
```

```
# In[49]:

# b
xi, yi = np.linspace(-3, 1, 100), np.linspace(0, 4, 100)
xi, yi = np.meshgrid(xi, yi)
miu = np.array([-1, 2])
cov = np.array([[2, 1], [1, 3]])
z = np.array([gaussian_2d(miu, cov, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]).re
CS = plt.contour(z, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(CS, inline=1, fontsize=10)
plt.axis('equal')
plt.show()


# In[53]:

# c
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
miu1 = np.array([0, 2])
cov1 = np.array([[2, 1], [1, 1]])
z1 = np.array([gaussian_2d(miu1, cov1, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
miu2 = np.array([2, 0])
cov2 = np.array([[2, 1], [1, 1]])
z2 = np.array([gaussian_2d(miu2, cov2, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
CS = plt.contour(z1-z2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(CS, inline=1, fontsize=10)
plt.axis('equal')
plt.show()


# In[55]:

# d
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
miu1 = np.array([0, 2])
cov1 = np.array([[2, 1], [1, 1]])
z1 = np.array([gaussian_2d(miu1, cov1, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
miu2 = np.array([2, 0])
cov2 = np.array([[2, 1], [1, 3]])
z2 = np.array([gaussian_2d(miu2, cov2, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())])
CS = plt.contour(z1-z2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(CS, inline=1, fontsize=10)
plt.axis('equal')
plt.show()
```

```
# In[56]:

# c
xi, yi = np.linspace(-3, 5, 100), np.linspace(-3, 5, 100)
xi, yi = np.meshgrid(xi, yi)
miu1 = np.array([1, 1])
cov1 = np.array([[2, 0], [0, 1]])
z1 = np.array([gaussian_2d(miu1, cov1, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]
miu2 = np.array([-1, -1])
cov2 = np.array([[2, 1], [1, 2]])
z2 = np.array([gaussian_2d(miu2, cov2, x, y) for (x,y) in zip(xi.ravel(), yi.ravel())]
CS = plt.contour(z1-z2, extent=[xi.min(), xi.max(), yi.min(), yi.max()])
plt.clabel(CS, inline=1, fontsize=10)
plt.axis('equal')
plt.show()


# ---
# ## Q3

# In[66]:

from numpy.random import normal
X1 = normal(3, 3, 100)
X2 = normal(4, 2, 100)
X2 += X1/2


# In[74]:

miu = np.array([X1.mean(), X2.mean()])
print("Mean: \n", miu)


# In[73]:

X = np.vstack((X1, X2))
cov = np.cov(X)
print("Covariance: \n", cov)


# In[79]:

w, v = np.linalg.eig(cov)
print("Eigenvalue: \n", w)
print("Eigenvectors: (each column) \n", v)
```

```
# In[109]:

v1 = v[:, 0].dot(w[0])
v2 = v[:, 1].dot(w[1])

plt.scatter(X1, X2)
plt.xlim([-15,15])
plt.ylim([-15,15])
ax = plt.axes()
ax.set_aspect('equal')
ax.arrow(miu[0], miu[1], v1[0], v1[1], head_width=1, head_length=1, fc='g', ec='g')
ax.arrow(miu[0], miu[1], v2[0], v2[1], head_width=1, head_length=1, fc='r', ec='r')
plt.show()


# In[123]:

X1c = X1 - miu[0]
X2c = X2 - miu[1]
Xr = v.T.dot(np.array([X1c, X2c]))
plt.scatter(Xr[0, :], Xr[1, :])
plt.xlim([-15,15])
plt.ylim([-15,15])
ax = plt.axes()
ax.set_aspect('equal')
plt.show()
```

2. Code for Problem 6:

```
# coding: utf-8

# ## CS189/289A HW3 code part2
# ### Yicheng Chen
# ### 02/15/2017

# In[67]:

get_ipython().magic('matplotlib inline')
import scipy.io
import random
from random import shuffle
import matplotlib.pyplot as plt
import numpy as np
from skimage.feature import hog
import pandas as pd
import cv2
```

```
# ---
# ## (a) fit Gaussian distribution to each digit class

# In[ ]:

mnist = scipy.io.loadmat('mnist/train.mat')


# In[ ]:

training_X = mnist['trainX'][:, :-1]
training_y = mnist['trainX'][:, -1]


# In[8]:

def normalize_row(X):
    Xn = np.zeros(X.shape)
    for i in range(X.shape[0]):
        x = X[i, :]
        Xn[i, :] = (x/(np.sqrt(x.dot(x))+1e-15))
    return Xn


# In[ ]:

training_Xn = normalize_row(training_X)


# In[ ]:

for digit in range(1, 2):
    training_digit = training_Xn[training_y==digit, :]
    mean = training_digit.mean(axis=0)
    cov = np.cov(training_digit.T)


# ---
# ## (b) Visualize the covariance matrix

# In[ ]:

plt.imshow(cov)
plt.colorbar()
plt.show()
```

```
# ---
# ## (c) Classify the digits in the test set on the basis of posterior probabilities wi

# ### (i) LDA

# In[14]:

class LDA:
    def __init__(self):
        self.Mean = 0
        self.Cov = 0
        self.P = 0

    def fit(self, X, y):
        N = len(y)
        labels = set(y)
        Cov = 0
        Mean = np.zeros([len(labels), X.shape[1]])
        P = np.zeros([len(labels)])
        for i, label in enumerate(labels):
            X_class = X[y==label, :]
            Nc = X_class.shape[0]
            P[i] = Nc / N
            Mean[i, :] = X_class.mean(axis=0)
            Cov += np.cov(X_class.T) * Nc
        Cov = Cov / N
        self.Mean = Mean
        self.Cov = Cov
        self.P = P

    def predict(self, X):
        PreMat = np.linalg.pinv(self.Cov)
        L = self.Mean.dot(PreMat).dot(X.T).T             - 1/2*np.diag(self.Mean.dot(Pr
        yp = np.argmax(L.T, axis=0)
        return yp

    def accuracy(self, X, y):
        yp = self.predict(X)
        N = len(y)
        Nerror = (yp != y).sum()
        return 1 - Nerror/N



# In[ ]:

mnist = scipy.io.loadmat('mnist/train.mat')
```

```
# In[ ]:

training = mnist['trainX']
shuffle(training)
validation = training[0:10000]
training = training[10000:]


# In[ ]:

training_X = normalize_row(training[:, :-1])
training_y = training[:, -1]
validation_X = normalize_row(validation[:, :-1])
validation_y = validation[:, -1]


# In[ ]:

subset_size = np.array([100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000])


# In[ ]:

all_accuracy = np.zeros(len(subset_size))


# In[ ]:

for i, N in enumerate(subset_size):
    idx = random.sample(range(50000), N)
    lda = LDA()
    lda.fit(training_X[idx, :], training_y[idx])
    all_accuracy[i] = lda.accuracy(validation_X, validation_y)


# In[ ]:

plt.plot(subset_size, all_accuracy * 100)
plt.xscale('log')
plt.xlabel('Training set size')
plt.ylabel('Accuracy(%)')
plt.show()


# ### (ii) QDA
```

```
# In[15]:

class QDA:
    def __init__(self, a):
        self.Mean = 0
        self.Cov = 0
        self.P = 0
        self.a = a

    def Q(self, X, m, C, p):
        PreMat = np.linalg.inv(C)
        Const = - 1/2*np.log(np.linalg.det(C)+1e-20) + np.log(p)
        Qc = np.array([(-1/2*(x-m).dot(PreMat).dot(x-m)) + Const for x in X])
        return Qc

    def fit(self, X, y):
        N = len(y)
        d = X.shape[1]
        labels = set(y)
        Cov = np.zeros([X.shape[1], d, len(labels)])
        Mean = np.zeros([len(labels), d])
        P = np.zeros([len(labels)])
        for i, label in enumerate(labels):
            X_class = X[y==label, :]
            Nc = X_class.shape[0]
            P[i] = Nc / N
            Mean[i, :] = X_class.mean(axis=0)
            Cov[:, :, i] = np.cov(X_class.T) + self.a*np.eye(d)
        self.Mean = Mean
        self.Cov = Cov
        self.P = P

    def predict(self, X):
        nC = len(self.P)
        L = np.zeros([nC, len(X)])
        for i in range(nC):
            L[i, :] = self.Q(X, self.Mean[i], self.Cov[:, :, i], self.P[i])
        yp = np.argmax(L, axis=0)
        return yp

    def accuracy(self, X, y):
        yp = self.predict(X)
        N = len(y)
        Nerror = (yp != y).sum()
        return 1-Nerror/N
```

```
# In[ ]:

subset_size = np.array([100, 200, 500, 1000, 2000, 5000, 10000, 30000, 50000])
all_accuracy = np.zeros(len(subset_size))


# In[ ]:

for i, N in enumerate(subset_size):
    idx = random.sample(range(50000), N)
    qda = QDA(1e-8)
    qda.fit(training_X[idx, :], training_y[idx])
    all_accuracy[i] = qda.accuracy(validation_X, validation_y)


# In[ ]:

plt.plot(subset_size, all_accuracy * 100)
plt.xscale('log')
plt.xlabel('Training set size')
plt.ylabel('Accuracy(%)')
plt.show()


# ### (iii) which perform better?

# (Written answer.) QDA

# ### (iv) Kaggle

# In[153]:

#Calculate the HOG feature
def hog_feature(data):
    data_hog = np.zeros(data.shape)
    for i in range(0, len(data)):
        feature = data[i, :]
        image = feature.reshape(28, 28)
        fd, hog_image = hog(image, orientations=8, pixels_per_cell=(4, 4), cells_per_bl
        data_hog[i, :] = hog_image.ravel()
    return data_hog


# In[154]:

mnist = scipy.io.loadmat('mnist/train.mat')
test = scipy.io.loadmat('mnist/test.mat')
training_X = mnist['trainX'][:, :-1]
```

```
training_y = mnist['trainX'][:, -1]
test_X = test['testX']


# In[155]:

training_Xh = normalize_row(hog_feature(training_X))
test_Xh = normalize_row(hog_feature(test_X))


# In[156]:

training_Xcombined = np.concatenate((normalize_row(training_X), training_Xh), axis=1)
test_Xcombined = np.concatenate((normalize_row(test_X), test_Xh), axis=1)


# In[157]:

std = np.std(training_Xcombined, axis=0)


# In[158]:

# use the max N variance word to build model
# max_idx = std.argsort()[:]
# training_X_max = training_Xcombined[:, max_idx]
# test_X_max = test_Xcombined[:, max_idx]


# In[159]:

lda = LDA()
lda.fit(training_Xcombined, training_y)

yp_lda = lda.predict(test_Xcombined)
lda.accuracy(training_Xcombined, training_y)


# In[65]:

df = pd.DataFrame({'Category': yp_lda})
df.index.rename('Id', inplace=True)
df.to_csv('mnist/mnist_predict_org+hog_lda.csv')


# ---
# ## (d) Spam
```

```
# In[2]:

spam = scipy.io.loadmat('spam/spam_data.mat')


# ### Bag-of-Words feature

# In[3]:

from sklearn.feature_extraction.text import CountVectorizer
import glob


# In[4]:

SPAM_DIR = 'spam/'
HAM_DIR = 'ham/'
TEST_DIR = 'test/'
NUM_TEST_EXAMPLES = 10000
spam_filenames = glob.glob('spam/' + SPAM_DIR + '*.txt')
ham_filenames = glob.glob('spam/' + HAM_DIR + '*.txt')
test_filenames = ['spam/' + TEST_DIR + str(x) + '.txt' for x in range(NUM_TEST_EXAMPLES


# In[5]:

all_text = []
for file in spam_filenames+ham_filenames: # use only training set data to build BOG
    with open(file, "r", encoding='utf-8', errors='ignore') as f:
        all_text.append(f.read())


# In[6]:

all_test_text = []
for file in test_filenames: # use only training set data to build BOG
    with open(file, "r", encoding='utf-8', errors='ignore') as f:
        all_test_text.append(f.read())


# In[9]:

vectorizer = CountVectorizer(min_df=4) # min word length=4
training_X = normalize_row(vectorizer.fit_transform(all_text).toarray())
test_X = normalize_row(vectorizer.transform(all_test_text).toarray())


# In[10]:
```

```
training_y = np.concatenate((np.ones(len(spam_filenames)), np.zeros(len(ham_filenames))


# In[11]:

std = np.std(training_X, axis=0)


# In[12]:

# use the max 2000 variance word to build model
max_idx = std.argsort()[-2000:]
training_X_max = training_X[:, max_idx]
test_X_max = test_X[:, max_idx]


# In[16]:

lda = LDA()
lda.fit(training_X_max, training_y)
lda.accuracy(training_X_max, training_y)


# In[17]:

# Kaggle submission
yp = lda.predict(test_X_max)
df = pd.DataFrame({'Category': yp})
df.index.rename('Id', inplace=True)
df.to_csv('spam/spam_BOW_predict.csv')


# ## (e) For Experts

# In[18]:

# max 10 variance words
max_idx = std.argsort()[-10:]
training_X_max = training_X[:, max_idx]
test_X_max = test_X[:, max_idx]


# In[19]:

lda = LDA()
lda.fit(training_X_max, training_y)
lda.accuracy(training_X_max, training_y)
```

```
# In[20]:

# min 10 variance words
min_idx = std.argsort()[:10]
training_X_min = training_X[:, min_idx]
test_X_min = test_X[:, min_idx]


# In[21]:

lda = LDA()
lda.fit(training_X_min, training_y)
lda.accuracy(training_X_min, training_y)


# In[ ]:
```