CSE 546 HW #1

Sam Kowash

November 8, 2018

(1) Gaussians

We explore bivariate Gaussian distributions with given means and covariance matrices

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \qquad \mu_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \qquad \mu_3 = \begin{bmatrix} 2 \\ -2 \end{bmatrix},$$
$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \qquad \Sigma_2 = \begin{bmatrix} 2 & -1.8 \\ -1.8 & 2 \end{bmatrix}, \qquad \Sigma_3 = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}.$$

For each of $\mathcal{N}(\mu_i, \Sigma_i)$, we drew 100 points $X_{i,j}$ and computed the sample means and covariance matrices. Each of the left-hand plots in Fig. 1.1 shows the points drawn, along with the eigenvectors $u_{i,1}$ and $u_{i,2}$ of the sample covariance originating at the sample mean $\hat{\mu}_i$ and scaled by the square roots of the corresponding eigenvalues $\lambda_{i,1}, \lambda_{i,2}$.

Each right-hand plot shows the same points centered and projected onto a scaled and rotated coordinate basis defined by

$$\tilde{X}_{i,j} \equiv \begin{bmatrix} \frac{1}{\sqrt{\lambda_{i,1}}} u_{i,1}^T (X_{i,j} - \hat{\mu}_i) \\ \frac{1}{\sqrt{\lambda_{i,2}}} u_{i,2}^T (X_{i,j} - \hat{\mu}_i) \\ \vdots \end{bmatrix}$$

This transformation shifts and decorrelates the drawn points to produce a zero-mean, identity-covariance distribution with respect to the new coordinate directions, analogous to the standardization of univariate normal distributions.

(2) MLE and bias–variance tradeoff

2. If we draw $x_1, \ldots, x_n \sim \text{uniform}(0, \theta)$ for some positive parameter θ, then

$$\mathbb{P}(x_1, \ldots, x_n \mid \theta) = \begin{cases} \left(\frac{1}{\theta}\right)^n & \text{if } x_1, \ldots, x_n \in [0, \theta] \\ 0 & \text{else} \end{cases}$$

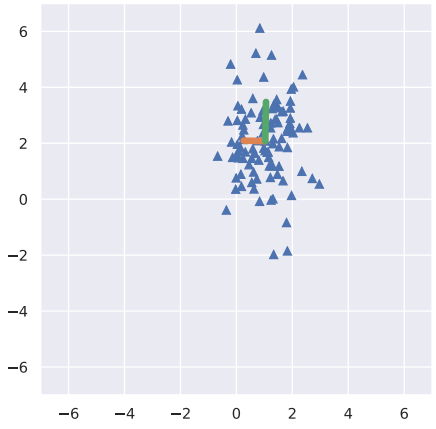
   We can see that the likelihood increases as θ decreases as long as all points drawn still lie in $[0, \theta]$, so we conclude that the MLE is $\hat{\theta} = \max(x_1, \ldots, x_n)$.

3. Let $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ be drawn from some population. We define

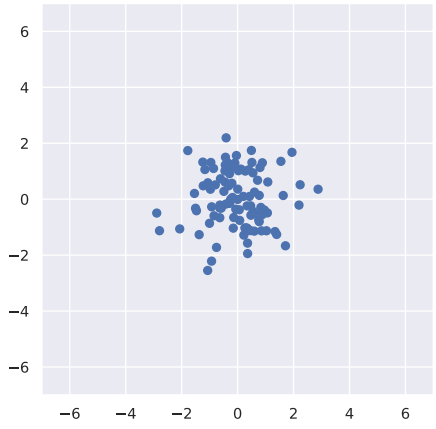
$$\hat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2.$$

   Let $(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_m, \tilde{y}_m)$ be test data drawn from the same population. We define the training and test losses

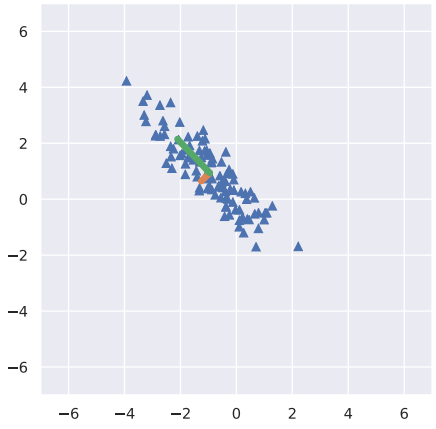
$$R_{\text{tr}} = \frac{1}{n} \sum_{i=1}^n \left(y_i - \hat{w}^T x_i\right)^2, \qquad R_{\text{te}} = \frac{1}{m} \sum_{i=1}^n \left(\tilde{y}_i - \hat{w}^T \tilde{x}_i\right)^2.$$



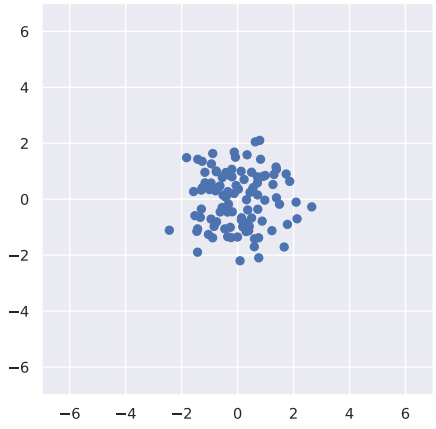
(a) Points and sample covariance eigenvectors drawn from $\mathcal{N}(\mu_1, \Sigma_1)$



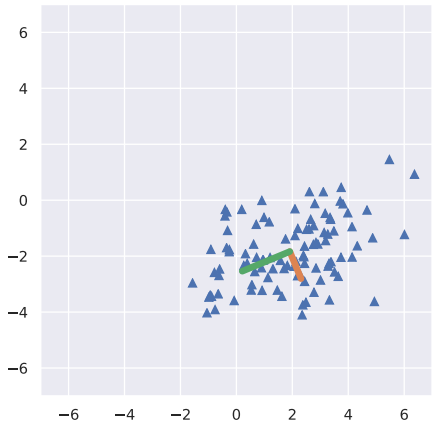
(b) Centered and rescaled points from $\mathcal{N}(\mu_1, \Sigma_1)$



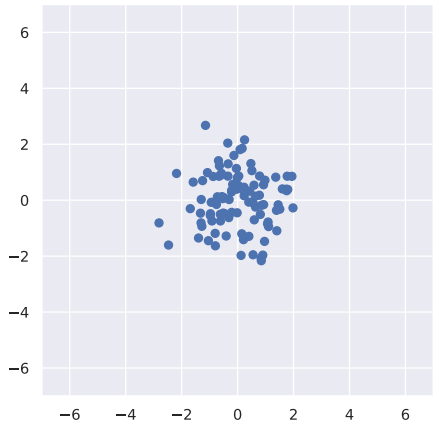
(c) Points and sample covariance eigenvectors drawn from $\mathcal{N}(\mu_2, \Sigma_2)$



(d) Centered and rescaled points from $\mathcal{N}(\mu_2, \Sigma_2)$



(e) Points and sample covariance eigenvectors drawn from $\mathcal{N}(\mu_3, \Sigma_3)$



(f) Centered and rescaled points from $\mathcal{N}(\mu_3, \Sigma_3)$

Figure 1.1: Illustrations of bivariate normal distributions and their standardizations

We are interested in comparing the expected values of these losses over all possible training and test draws. Note that because our points are drawn i.i.d., each term in R_{tr} should make the same expected contribution and each term in R_{te} should make the same expected contribution (this is to say that in the expectation, $\hat{w}$ should not "prefer" any particular data direction). Thus without loss of generality we can analyze the case $m = n$. ...

(3) Ridge regression on MNIST

5. We develop a least-squares classifier for the MNIST handwritten digit set.

   (a) We want to minimize the objective

   $$\widehat{W} = \underset{W \in \mathbb{R}^{d \times k}}{\arg\min} \sum_{i=0}^{n} \|W^T x_i - y_i\|_2^2 + \lambda \|W\|_F^2,$$

   but have run out of time to show that the answer is $(X^T X + \lambda I_d)^{-1} X^T Y$.

   (b) Nothing reportable in this part.

   (c) Using the pixels directly as features, we obtained a test error of 13.97%. Code not included because it's a boring subset of the code that is included.

   (d) We obtained a better feature set through the transform

   $$h(x) = \cos(Gx + b),$$

   where G is a $p \times d$ matrix with entries drawn i.i.d. from $\mathcal{N}(0, 0.1)$ and b is a length p vector drawn from a uniform distribution on $[0, 2\pi)$. We applied single-sample cross-validation with an 80/20 split to optimize the selection of p, measuring training and validation errors for 371 values ranging from 1 to 23000. The results are displayed in Fig. 3.1.

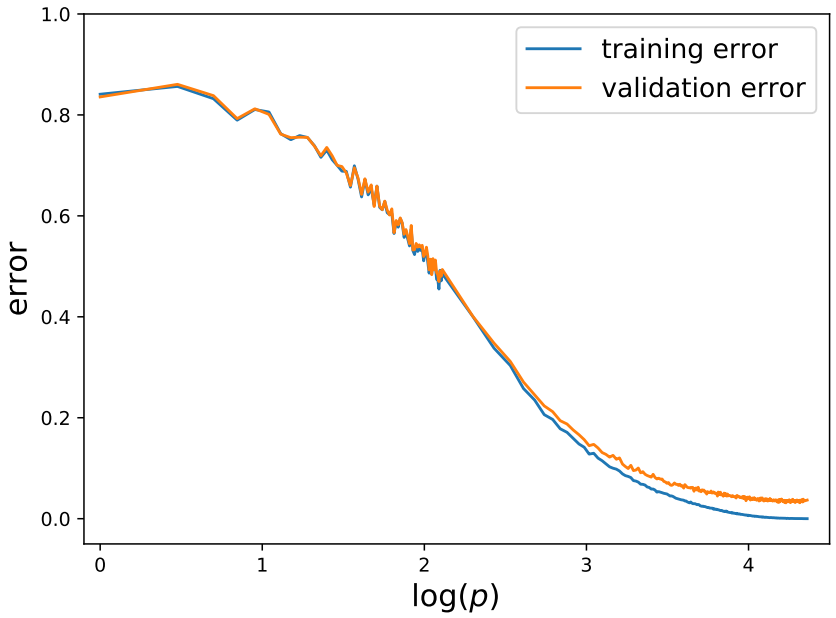


Figure 3.1: Training and validation errors for various values of p. (Apparent variation in noise scale is due partly to log scale and different sampling densities in different regions.)

There was no noticeable absolute minimum of validation error, and time limitations precluded exploring higher p with useful resolution. (Even on Hyak, any $p > 20000$ can take ∼20–30 minutes to train.) Accordingly we took $p = 10000$, where the validation error seems to begin leveling off, as our final value, since higher p increased computation costs with little appreciable improvement in validation accuracy.

(e) We can use Hoeffding's inequality to establish a 95% confidence interval for the test error. For a family of m bounded random variables X_i with expected mean μ — in this case, the 0/1 loss for each test data point, whose expected mean is the true classification error — it gives that for $\delta \in (0, 1)$,

$$\mathbb{P}\left(\left|\left(\frac{1}{m}\sum_{i=1}^{m} X_i\right) - \mu\right| \geqslant \sqrt{\frac{\log(2/\delta)}{2m}}\right) \leqslant \delta. \tag{3.1}$$

In our case we take $\delta = 0.05$ and $m = 10000$ (the size of the test set) and find that with at least 95% probability, the true error lies within 0.0136 of our measured error. Using $p = 10000$ as determined above, we measure a test error of 3.99%, so our 95% confidence interval is $[2.63, 5.35]$.

```
#!/usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mnist import MNIST
import sys
import time

def load_test():
    mndata = MNIST('../../mnist/data/')
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_test = X_test/255.

    return X_test, labels_test

def load_train():
    mndata = MNIST('../../mnist/data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_train = X_train/255.

    return X_train, labels_train

def demean(X): #array must have more than one axis!
    mu = np.mean(X, axis=0)
    return X-mu, mu

def pred_error(labels_pred, labels_true):
    if len(labels_pred) != len(labels_true):
        print("Unequal label list lengths.")
        return

    misses = np.count_nonzero(labels_pred-labels_true)
    total = len(labels_true)
    print("Missed %s out of %s, relative error %s." %(misses,total,misses/total))

    return misses/total


def train(X, Y, L): # X is (n,d), Y is (n,k), L is reg. constant
    if len(X) != len(Y):
        print("Unequal input lengths; what have you done?")
        return

    d,k,n = len(X[0]), len(Y[0]), len(X)

    Xt = np.transpose(X)

    M = np.dot(Xt,X)
```

```
    N = np.dot(Xt,Y)

    return np.linalg.solve(M+L*np.identity(d), N)


def predict(W,X,Y_mu=0): # W is (d,k), X is (m,d), Y_mu is (1,k)
    m = len(X)

    Y_pred = np.dot(X, W) + Y_mu

    return np.argmax(Y_pred, axis=1)

def cos_tx(p,d):
    var = 0.1
    Gt = np.random.randn(d,p)*np.sqrt(var)
    b = np.random.uniform(0,2*np.pi,p)


    return lambda X: np.cos(np.dot(X,Gt) + b)


#command-line signature: mnist_ridge.py p
p = int(sys.argv[1])



# load data and labels
X_train_tot, labels_train_tot = load_train()
# X_test, labels_test = load_test()

# encode labels as unit vectors
codes = np.identity(10)
Y_train_tot = np.array([codes[labels_train_tot[i]]
                        for i in range(len(labels_train_tot))])
# Y_test = np.array([codes[labels_test[i]] for i in range(len(labels_test))])



# partition data into train/val
idx = np.random.permutation(60000)
ind_train = idx[0:48000]
ind_val = idx[48000:]

X_train = X_train_tot[ind_train]
Y_train = Y_train_tot[ind_train]
labels_train = labels_train_tot[ind_train]

X_val = X_train_tot[ind_val]
```

```
Y_val = Y_train_tot[ind_val]
labels_val = labels_train_tot[ind_val]




# map features
feature_tx = cos_tx(p, len(X_train[0]))
H_train = feature_tx(X_train)
H_val = feature_tx(X_val)
# H_test = feature_tx(X_test)

# H_train, H_mu = demean(H_train)
# Y_train, Y_mu = demean(Y_train)

print("Created feature matrix of shape %s" %str(H_train.shape))

# train classifier on de-meaned features
W = train(H_train, Y_train, 1e-4)
print("Trained classifier of shape %s with Frobenius norm %s" %(str(W.shape), np.tr


# predict to measure train/validation/test error
labels_train_pred = predict(W, H_train)
labels_val_pred = predict(W, H_val)
# labels_test_pred = predict(W, H_test)

train_error = pred_error(labels_train_pred, labels_train)
val_error = pred_error(labels_val_pred, labels_val)
# test_error = pred_error(labels_test_pred, labels_test)


with open(f'data/{p}','a') as f:
    f.write('%s,%s,%s\n' %(p,train_error,val_error))
```

```
#!/usr/bin/env python
import numpy as np
from mnist import MNIST
import sys

def load_test():
    mndata = MNIST('../../mnist/data/')
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_test = X_test/255.

    return X_test, labels_test

def load_train():
    mndata = MNIST('../../mnist/data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_train = X_train/255.

    return X_train, labels_train

def demean(X): #array must have more than one axis!
    mu = np.mean(X, axis=0)
    return X-mu, mu

def pred_error(labels_pred, labels_true):
    if len(labels_pred) != len(labels_true):
        print("Unequal label list lengths.")
        return

    misses = np.count_nonzero(labels_pred-labels_true)
    total = len(labels_true)
    print("Missed %s out of %s, relative error %s." %(misses,total,misses/total))

    return misses/total


def train(X, Y, L): # X is (n,d), Y is (n,k), L is reg. constant
    if len(X) != len(Y):
        print("Unequal input lengths; what have you done?")
        return

    d,k,n = len(X[0]), len(Y[0]), len(X)

    Xt = np.transpose(X)

    M = np.dot(Xt,X)
    N = np.dot(Xt,Y)

    return np.linalg.solve(M+L*np.identity(d), N)
```

```
def predict(W,X,Y_mu=0): # W is (d,k), X is (m,d), Y_mu is (1,k)
    m = len(X)

    Y_pred = np.dot(X, W) + Y_mu

    return np.argmax(Y_pred, axis=1)

def cos_tx(p,d):
    var = 0.1
    Gt = np.random.randn(d,p)*np.sqrt(var)
    b = np.random.uniform(0,2*np.pi,p)


    return lambda X: np.cos(np.dot(X,Gt) + b)




# load data and labels
X_train, labels_train = load_train()
X_test, labels_test = load_test()


#command-line signature: mnist_ridge.py p
p = int(sys.argv[1])
d = X_train.shape[1]
m = X_test.shape[0]
delta = 0.05

# encode labels as unit vectors
codes = np.identity(10)
Y_train = np.array([codes[labels_train[i]]
                    for i in range(len(labels_train))])
Y_test = np.array([codes[labels_test[i]] for i in range(len(labels_test))])


feature_tx = cos_tx(p,d)

H_train = feature_tx(X_train)
H_test = feature_tx(X_test)

H_train, H_mu = demean(H_train)
Y_train, Y_mu = demean(Y_train)

W = train(H_train, Y_train, 1e-4)
```

```
labels_pred = predict(W, H_test-H_mu, Y_mu)
test_error = pred_error(labels_pred,labels_test)
conf_int = np.sqrt(np.log(2/delta)/(2*m))

print("With p=%s, the test error is %3f +/- %3f." %(p,test_error,conf_int))
```