sta314 HW4

Shimon Nauenberg

1 Q1

1.1 a

as an important note we are using the notation k^i to refers to the i'th data point Let us being by fixing $k \in K$. Now let us notice that since $\sum_{k \in K} \theta_k = 1$, then we can define

$$\theta_k = 1 - \sum_{\substack{q \in K \\ q \neq k}} \theta_q$$

. Now let us notice that since we are interested in solving for $\hat{\theta_k}$ then we are interested in in solving for the MLE for θ_k. now let us consider the joint distribution $p(\Theta, \mathbf{X})$, and assume for the sake of computation that $x \in \mathbf{X}$ are independent given Θ. Therefore, we are able to notice that

$$P(\Theta, \mathbf{X}) = \mathbf{p(\Theta)p(x_1 \mid \Theta) \cdots p(x_k \mid \Theta)}$$

by the naive bayes assumption. Since we are interested in solving for θ_k, and we have determined a relationship between θ_k and $\theta_{r \neq k}$, then we should be able to treat this similarly to our previous problem that we discussed in lecture with spam detection. We can define a class t such that if $x = k$ then $t = 1$ otherwise $t = 0$, and therefore $p(t = 1) = \theta_k$. Now let us redefined our joint distribution as $P(t, x_1, \cdots, x_k)$ and notice that this is equivalent to our previous distribution outlined above for our purposes as we are only interested in θ_k. Now we can notice that again by the naive bayes assumption we can again split this up into $p(t)p(x_1 \mid t) \cdots p(x_k \mid t)$. Now let us notice that we can consider $\hat{t}$ and find it's MLE which would be equivalent of our desired MLE. Now let us notice therefore that we would end up with the following

$$\begin{aligned}
\ell(\theta) &= \sum_N \log p(t^i, x^i) && (1)\\
&= \sum_N \log p(x^i \mid t^i) p(t^i) \text{ per joint definition} && (2)\\
&= \sum_{i \in N} \log \left[p(t^i) \prod_{j \in K} p(x_j^i \mid t^i) \right] \text{ by naive bayes} && (3)\\
&= \sum_{i \in N} \left[\log p(t^i) + \sum_{j \in K} \log p(x_j^i \mid t^i) \right] \text{ by distributing the log function} && (4)\\
&= \sum_N log p(t^i) + \sum_{j \in k} \sum_{i \in N} log p(x_j^i \mid t^i) \text{ by distributing the summation} && (5)
\end{aligned}$$

Now since we are interested in finding the mle of the class probabilities of θ_k then we are interested in solving for the MLE for the first term in (5). Now let since we have already declared that $p(t = 1) = \theta_k$, then $P(t = 0) = 1 - (\theta_k)$, and therefore, $p(t^i) = \theta_k^{t^i}(1 - \theta_k)^{1 - t^i}$.Therefore our log likelihood for the first term can be split up into two terms $f(\theta_k, t, N) = \sum_N t^i log(\theta_k) + \sum_N (1 - t^i) log(1 - \theta_k)$.

Now let us take the derivative of this WRT to θ_k and set it to zero and solve for the MLE. Therefore,

$$\partial_{\theta_k} f = \frac{1}{\theta_k} \sum_N t^i - \frac{1}{1 - \theta_k} \sum_N (1 - t^i) = 0$$

Now to solve for the θ_k we get

$$\theta_k = \frac{\sum_N t^i}{\sum_N t^i + \sum_N (1 - t^i)} = \frac{\sum_N t^i}{N}$$

. Now since we treated $t = 1$ when $x = k$ the numerator is equivalent to $\mathbb{1}[t^i = 1]$. now to relate this back to our original problem. If we notice that when x_i has the label of $t^i == \theta_k$ then this is the same as $x_k^i == 1$ Which would be the same as $\frac{N_k}{N}$, the fraction of labeld k over the entire dataset.

Now since we want to ensure that this is a global maximum we have to ensure that our solution is positive let us notice that $\partial_{\theta_k}^{(2)} f = -(\frac{\sum_N t^i}{\theta_k^2} + \frac{\sum_N (1 - t^i)}{(1 - \theta_k)^2})$. Now let us ntoice that $\frac{\sum_N t^i}{\theta_k^2} \geq 0$ and that $\frac{\sum_N (1 - t^i)}{10\theta_k)^2} \geq 0$ which implies that the second derivative is negative, implying that our solution must be a global maximum as desired. Note, we know that it must be negative and non zero as at least one of the two terms in the sum must be strictly positive by our definition of t, and hence the netire thing is negative. .

1.2 b

Let us notice that by bayes theorem we know that

$$p(\Theta \mid X) \propto p(\Theta)p(X \mid \Theta)$$

. Now since we are assuming that

$$p(\Theta) \propto \theta_1^{a_1-1} \cdots \theta_K^{a_k-1}$$

. Now since we are assuming the naive bayes assuming we are able to know that

$$\begin{aligned}
P(\Theta \mid X) &\propto p(\Theta, X)p(X) \text{by definition of joint} \\
&\propto p(\Theta) \prod_K p(x_i \mid \Theta) \text{by naive bayes} \\
&\propto \prod_k \theta_i^{a_i-1} \prod_{i \in K} \prod_{j \in N} \theta_i^{x_i^j} \text{ by expanding out the hint 2 in the previous question} \\
&\propto \prod_k \theta_k \prod_k \theta_k^{\sum_j x_i^j} \text{by exponent rules} \\
&\propto \prod_k \theta_k^{a_k-1+\sum_{i \in N} x_k^i} \text{ by joining like terms and exponent rules}
\end{aligned}$$

Now we can notice that this is again a product of k terms which again results ina a dirichelt distirubution

1.3 c

We are interested in solving for Θ, what we will do is solve for each $\theta_k{}_{k=1}^{K}$ and from there be able to extrapolate that solving each component of Θ will give us the maximum for Θ. Since we know that we are dealing with a dirchlet posterior distribution then we know that we can do as follows. First let us fix $k \in 1 \cdots K$. Then we are able to solve as follows

$$\Theta_{MAP} = argmax_{\theta} p(\Theta \mid X) \tag{6}$$
$$= logp(\Theta) + logp(X \mid \Theta) \tag{7}$$

Now since since we are intersted ub solving for θ_k then we could do as follows, since we know we are dealing with a dirichlet distribution then we know that we could treat this as follows/

$$r \in \mathbb{R} + (a_k - 1)log(\theta_k) + \sum_{i \neq k}(a_i - 1)log(1 - \theta_k) + N_k log(\theta_k) + \sum_{i \neq k} N_i log(1 - \theta_k) \tag{8}$$

Note firs that by the nature of x^i being 1 of k encoding we can replace x^i with N_i Now if we take the derivative w.r.t to each θ_k and equate it to zero, we end up with

$$\frac{a_k - 1}{\theta_k} - \frac{\sum_{i \neq k}(a_i - 1)}{1 - \theta_k} + \frac{N_k}{\theta_k} - \frac{N - N_k}{1 - \theta_k} = 0 \tag{9}$$

which gives us the invevitable values for $\theta_{kmap} = \frac{a_k - 1 + N_k}{\sum_i (a_i - 1) + N}$

Therefore, if we generalize this for Θ We end up with the following, let $a = (a_1, \cdots a_k)$, and let $N_K = (N_1, \cdots N_k)$

$$\hat{\Theta}_{map} = \frac{a - 1 + N_K}{\sum_i (a_i - 1) + N} \tag{10}$$

Notice that since we are able to assume that $a_i > 1$, then we know that the denominator is well defined and non-zero. Further we also know that since Θ are probabilites and must be non zero we also know that the $\hat{\Theta_{map}}$ will also be positive.

1.4 d

Let us begin by finding the distribution of $p(X^{N+1} \mid D)p(\Theta \mid D)$. We can notice by the previous question that the second element in the product is has a distribution of

$$\prod_K \theta_k^{a_k - 1 + \sum_{i \in D} x_k^i}$$

and we can notice by the assumption laid out in the question that the distribution of the first term is

$$\prod_k \theta_k^{x_k^{N+1}}$$

.

Let $p(X^{n+1} = 1 \mid D)$ be the conditional probability of X^{n+1} being catagorized as catagory k. Therefore, we can notice that our product is distributed as

$$\prod_k \theta_k^{a_k - 1 + \sum_{i \in D \cup X^{N+1}} x_k^i}$$

. Therefore, we are able to use the hint to realize

$$\int p(x^{N+1} \mid \Theta) p(\Theta \mid D) D\Theta = E(\Theta) \tag{11}$$

$$E[\theta_k] = \frac{a_k + \sum_{i \in D \cup X^{N+1}} x_k^i}{\sum_k \left[a_k + \sum_{i \in D \cup X^{N+1}} x_k^i\right]} \tag{12}$$

Now if we notice that we are interested in $p(X^{n+1} = 1 \mid D)$, then we can notice that we arre only interested in solving for $E[\theta_k]$ as all the other expected values in the means vector will be irrelevant and have a zero value for our purposes. Therefore be know that

$$p(X^{N+1} = 1 \mid D) = \frac{a_k + \sum_{i \in D \cup X^{N+1}} x_k^i}{\sum_k \left[a_k + \sum_{i \in D \cup X^{N+1}} x_k^i\right]} \tag{13}$$

$$= \frac{1}{\sum_{\mathrm{J} \neq k} \left[a_j + \sum_{i \in D \cup X^{N+1}} x_j^i\right]} \tag{14}$$

as we are only interested in $E[\theta_k]$. Note since the expected value is defined as an integral over the entire domain for conitnues Rv's, then we can use the expected value to computer this integral as desired.

2 Q2

2.1 1

Let us being by finding a solution to p(x). We can use the law of total probability to determine an expresiion for p(x). **note eq(.2, t = i) refers to the equation in the homework**

$$p(x) = \sum_k p(x \mid t = k)p(t = k) \tag{15}$$

$$= \sum_k a_k p(x \mid t = k) \tag{16}$$

$$= \sum_k a_k (\prod_d 2\pi\sigma_d^2)^{(-.5)} \exp\big[- \sum_d \frac{1}{2\sigma_d^2}(x_d^k - \mu_{kd})^2\big] \tag{17}$$

Now we know that by the definition of conditional probability that

$$p(t = k \mid x) = \frac{p(x \mid t = k)p(t = k)}{p(x)} \tag{18}$$

Therefore, we know by eq15 and eq(.2), eq(.1), for the joint distribution we end up with $a_k \times eq(0.2, t = k)/eq(15)$ for our joint distribution. Therefore, to find an expression for eq(16), we end up with

$$p(t = k \mid x) = \frac{a_k eq(.2, t = k)}{\sum_i a_i eq(.2, t = i)} \tag{19}$$

Which result in

$$p(t = k \mid x) = \frac{1}{\sum_{i \neq k}(a_i eq(.2, t = i)} \tag{20}$$

$$= \frac{1}{\sum_{\imath \neq k} a_i (\prod_d 2\pi\sigma_d^2)^{(-.5)} \exp\big[- \sum_d \frac{1}{2\sigma_d^2}(x_d^i - \mu_{kd})^2\big]} \tag{21}$$

Now we can notice that this a sum of normals is also a normal, then we know that this resulting distribution is an inverse normal distribution. [1]

[1] I learned about inverse normals from this page

2.2 2

We can being by noticing that since each (t^i, X^i) is independent then we know that we know we can factor this into a product of each $(t^i X^i)$.

$$\ell(\Theta) = \prod_{i \to N} p(t^i) p(x^i \mid t^i) \text{ by joint distirubtion definition and factoring} \tag{22}$$

$$= \sum_{i \in N} \left[log(a_{t^i}) - \sum_d \frac{1}{2\sigma_d^2} (x_d^i - \mu_{t^i d})^2 + .5 log(\sigma_d^2)) \right] + C \text{ by applying the distribution} \tag{23}$$

this is after simplifying and applying the conditional distribution, the C is a constant

$$= \sum_i log(a_{t^i}) - \sum_i \sum_d \frac{1}{2\sigma_d^2} (x_d^i - \mu_{t^i d})^2 + .5 log(\sigma_d^2)) + C \tag{24}$$

2.3 3

Let us begin by noticing that we are only concerned with the second term, when solving for $\partial\sigma_d^2$ from equation 18 as it is hte only term containing sigma.

Now let us notice that this results in the following result,

$$\partial_{\sigma_d^2} = \sum_i [\frac{-1}{\sigma_d^4}(x_d^i - \mu_{id)})^2 + \frac{1}{\sigma_d^2}] = 0 \tag{25}$$

Therefore, we end up with the following solution,

$$\hat{\sigma_d}^2 = \frac{\sum_i (x_d^i - \mu_{id})^2}{N}$$

.

Now to consider $\partial_{\mu_{kd}}$Let us notice that our equation 18 again only concerns itself for the second term in our case as it's the only one containing μ. However, we should notice that our term contains $\mu_{t^i d}$ and not our desired μ_{kd}. In order to fix this we can do the following, instead of summming over all i, we can take the following sum

$$\sum_{\substack{i \\ t^i = k}}$$

, which would mean that we would only be considering x_d^i where $t^i = k$. Let us notice that this is justified as when we take the derivative, any term in the sum in hwich $t^i \neq k$, the derivative will be zero.

Therefore, we can notice that we end up with the following as our derivative

$$\sum_{\substack{i \\ t^i = k}} \frac{1}{\sigma_d^2}(x_d^i - \mu_{kd}) = 0$$

, Which implies that our optimal solution for

$$\hat{\mu}_{kd} = \sum_{\substack{i \\ t^i = k}} \frac{x_d^i}{N_k}$$

3 Q3

3.1 1

```
'''
HW4 Q3

Implement and evaluate the Conditional Gaussian classifier.
'''

import data
import numpy as np
import scipy.special
import matplotlib.pyplot as plt
from math import pi
def compute_mean_mles(train_data, train_labels):
    '''
    Compute the mean estimate for each digit class. You may iterate over
    the possible digits (0 to 9), but otherwise make sure that your code
    is vectorized.

    Arguments
        train_data: size N x 64 numpy array with the images
        train_labels: size N numpy array with corresponding labels

    Returns
        means: size 10 x 64 numpy array with the ith row corresponding
               to the mean estimate for digit class i
    '''
    # Initialize array to store means
    means = np.zeros((10, 64))
    train_labels = np.expand_dims(train_labels, axis = 1)
    # joines the axis
    joined_data = np.concatenate((train_data, train_labels), axis = 1)
    for i in list(range(0,9 + 1)):
        ith_data_with_y = joined_data[joined_data[:, -1] == i]
        ith_data = np.delete(ith_data_with_y, -1, axis = 1)
        m = np.mean(ith_data, axis = 0)
        means[i] = m

    # == YOUR CODE GOES HERE ==
    # ====
    return means

def compute_sigma_mles(train_data, train_labels):
    '''
    Compute the covariance estimate for each digit class. You may iterate over
    the possible digits (0 to 9), but otherwise make sure that your code
    is vectorized.

    Arguments
        train_data: size N x 64 numpy array with the images
        train_labels: size N numpy array with corresponding labels

    Returns
        covariances: size 10 x 64 x 64 numpy array with the ith row corresponding
```

```
            to the covariance matrix estimate for digit class i
    '''
    # Initialize array to store covariances
    covariances = np.zeros((10, 64, 64))
    means = compute_mean_mles(train_data, train_labels)
    train_labels = np.expand_dims(train_labels, axis = 1)
    # joines the axis
    joined_data = np.concatenate((train_data, train_labels), axis = 1)
    for i in list(range(0, 9 + 1)):
        ith_data_with_y = joined_data[joined_data[:, -1] == i] # filter for label i

        ith_data = np.delete(ith_data_with_y, -1, axis = 1) # remove the label
        n,d = np.shape(ith_data)
        m = np.expand_dims(means[i], axis = 1)
        I = np.expand_dims(np.ones(n), axis = 1)
        first = (ith_data - I@m.T).T
        second = ith_data - I @ m.T
        cov = first @ second/n
        cov_I = 0.01 * np.ones(np.shape(cov))
        cov = cov + cov_I # for stability
        covariances[i] = cov


    # == YOUR CODE GOES HERE ==
    # ====
    return covariances

def generative_likelihood(digits, means, covariances):
    '''
    Compute the generative log-likelihood log p(x|t). You may iterate over
    the possible digits (0 to 9), but otherwise make sure that your code
    is vectorized.

    Arguments
        digits: size N x 64 numpy array with the images
        means: size 10 x 64 numpy array with the 10 class means
        covariances: size 10 x 64 x 64 numpy array with the 10 class covariances

    Returns
        likelihoods: size N x 10 numpy array with the ith row corresponding
                to logp(x^(i) | t) for t in {0, ..., 9}
    '''
    N = digits.shape[0]
    likelihoods = np.zeros((N, 10))
    d = digits.shape[1]
    for t in list(range(0,9 + 1)):
        m = np.expand_dims(means[t], axis = 1)
        c = covariances[t]
        det = np.linalg.det(c)
        inv = np.linalg.inv(c)
        first = -d/2*np.log(2*pi)
        second = -.5*np.log(det)
        I = np.expand_dims(np.ones(N), axis = 1)
        third = (digits -  m.T)
```

```
        fourth = (digits - m.T).T
        # this gives us our desired matrix, all we have to do is consider how
        #matrix multiplication owrks in order to get (x^i - \mu_k)@inv@(x^i - mu_k)T
        fifth = -.5*np.expand_dims((third@inv@fourth).diagonal(), axis = 1)
        ll = (first + second + fifth)
        likelihoods[:, t] = np.squeeze(ll) # squueze helps with formating,
        # does nothign to the data

    # == YOUR CODE GOES HERE ==
    # ====
    return likelihoods


def conditional_likelihood(digits, means, covariances):
    '''
    Compute the generative log-likelihood log p(t|x). Make sure that your code
    is vectorized.

    Arguments
        digits: size N x 64 numpy array with the images
        means: size 10 x 64 numpy array with the 10 class means
        covariances: size 10 x 64 x 64 numpy array with the 10 class covariances

    Returns
        likelihoods: size N x 10 numpy array with the ith row corresponding
                to logp(t | x^(i)) for t in {0, ..., 9}
    '''

    N = digits.shape[0]
    likelihoods = np.zeros((N, 10))
    first = np.log(10)
    gll = generative_likelihood(digits, means, covariances)
    second = -scipy.special.logsumexp(gll/10, axis = 1)
    # special scipy function, helps with numerical stability
    for t in list(range(0, 9 + 1)):
        third = gll[:,t]
        ll =  first + second + third
        likelihoods[:, t] = np.squeeze(ll)
    return likelihoods

def classify_data(digits, means, covariances):
    '''
    Classify new points by taking the most likely posterior class.
    Make sure that your code is vectorized.

    Arguments
        digits: size N x 64 numpy array with the images
        means: size 10 x 64 numpy array with the 10 class means
        covariances: size 10 x 64 x 64 numpy array with the 10 class covariances

    Returns
        pred: size N numpy array with the ith element corresponding
                to argmax_t log p(t | x^(i))
    '''
```

```
    genl = conditional_likelihood(digits, means, covariances)
    N = digits.shape[0]
    pred = np.zeros((N, 1))
    max_values = np.amax(genl, axis = 1)
    # this gives the max along each row
    for t in list(range(0, 9 +1)):
        genlt = genl[:,t]
        # diagonal ensures we only compare in the same column, otherwise it compares
        # each possible pair of elements
        pred[:,0] = np.where(genlt == max_values, t, pred).diagonal()




    return pred

def avg_conditional_likelihood(digits, labels, means, covariances):
    '''
    Compute the average conditional likelihood over the true class labels

        AVG( log p(t^(i) | x^(i)) )

    i.e. the average log likelihood that the model assigns to the correct class label.

    Arguments
        digits: size N x 64 numpy array with the images
        labels: size N x 10 numpy array with the labels
        means: size 10 x 64 numpy array with the 10 class means
        covariances: size 10 x 64 x 64 numpy array with the 10 class covariances

    Returns
        average conditional log-likelihood.
    '''
    cond_likelihood = conditional_likelihood(digits, means, covariances)

    # Compute as described above and return
    assert len(digits) == len(labels)
    sample_size = len(digits)
    total_prob = 0
    for i in range(sample_size):
        total_prob += cond_likelihood[i][int(labels[i])]

    return total_prob/sample_size



def main():
    train_data, train_labels, test_data, test_labels = data.load_all_data()

    # Fit the model
    means = compute_mean_mles(train_data, train_labels)
    covariances = compute_sigma_mles(train_data, train_labels)

    b = conditional_likelihood(train_data, means, covariances)
```

```
    # Evaluation
    train_log_llh = avg_conditional_likelihood(train_data, train_labels, means, covariances
    test_log_llh = avg_conditional_likelihood(test_data, test_labels, means, covariances)

    print('Train average conditional log-likelihood: ', train_log_llh)
    print('Test average conditional log-likelihood: ', test_log_llh)

    train_posterior_result = classify_data(train_data, means, covariances)
    test_posterior_result = classify_data(test_data, means, covariances)

    # this was changed so that we are only comparing equivalent entries
    train_accuracy = np.mean((train_labels.astype(int) == train_posterior_result).diagonal
    test_accuracy = np.mean((test_labels.astype(int) == test_posterior_result).diagonal())

    print('Train posterior accuracy: ', train_accuracy)
    print('Test posterior accuracy: ', test_accuracy)

    for i in range(10):
        (e_val, e_vec) = np.linalg.eig(covariances[i])
        # In particular, note the axis to access the eigenvector
        curr_leading_evec = e_vec[:,np.argmax(e_val)].reshape((8,8))
        plt.subplot(3,4,i+1)
        plt.imshow(curr_leading_evec, cmap='gray')
    plt.show()

if __name__ == '__main__':
    main()
```

3.2 2

Train average conditional log-likelihood: 39.14613852552519
Test average conditional log-likelihood: 34.78136184959013
Train posterior accuracy: 0.9812857142857143
Test posterior accuracy: 0.9605

Note we changed the mean calculation in order to compare only the like elements when calculating the accuracy, the current formula calculates every pair of two elements. Note, if we use the original formula we get this for our accuracy.
Train posterior accuracy: 0.1
Test posterior accuracy: 0.1