EECS-553 Homework 1

Vipransh Sinha

February 25, 2025

1. Maximum Likelihood Estimation

Consider a random variable $\mathbf{X}$ (possibly a vector) whose distribution (density function or mass function) belongs to a parametric family. The density or mass function may be written $f(\mathbf{x};\theta)$, where θ is called the parameter, and can be either a scalar or vector. For example, in the Gaussian family, θ can be a two-dimensional vector consisting of them mean and variance. Suppose the parametric family is known, but the value of the parameter is unknown. It is often of interest to estimate this parameter from realizations of $\mathbf{X}$. Maximum likelihood estimation is one of the most important parameter estimation techniques. Let $\mathbf{X_1}, \ldots, \mathbf{X_n}$ be iid (independent and identically distributed) random variables distributed according to $f(\mathbf{x};\theta)$. By independence, the joint distribution of the observations is the product

$$\prod_{i=1}^{n} f(x_i;\theta).$$

Viewed as a function of θ, this quantity is called the *likelihood* of θ. Because many common pdfs/pmfs have an exponential form, it is often more convenient to work with the *log-likelihood*,

$$\sum_{i=1}^{n} \log f(\mathbf{x_i};\theta).$$

A maximum likelihood estimator (MLE) of θ is a function $\hat{\theta}$ of the observed values $\mathbf{x_1}, \ldots, \mathbf{x_n}$ satisfying

$$\hat{\theta} \in \arg\max_{\theta} \sum_{i=1}^{n} \log f(\mathbf{x_i};\theta)$$

where arg max denotes the set of all values achieving the maximum. If there is always a unique maximizer, it is called the maximum likelihood estimator.

Negative Binomial Distribution The negative binomial distribution models the number of failures in a sequence of iid Bernoulli trials before r successes to take place. In particular, suppose $\{y_i\}_{i=1}^{m}$ is a sequence of random variables taking values 0 and 1, drawn iid from

Bernoulli(θ), with "success probability" θ, where 1 corresponds to "success" and 0 corresponds to "failure". The length m of the sequence is exactly the number of trials required to draw r success, i.e., we keep drawing a sample from Bernoulli(θ) until we obtain r successes and the number of draws it takes is m. So $m - r$ is the number of failures before the r^{th} success, which is a negative binomial random variable, and we write $(m - r) \sim^{iid} NB(r, \theta)$. Note that the θ is the same in NB(r,θ), and Bernoulli(θ), since the negative binomial distribution implies an underlying Bernoulli distribution.
The negative binomial distribution can be used to answer questions like: If I repeatedly flip a coin, what is the probability that the 5^{th} heads occurs on the 12^{th} flip?

(a) Let $X_1, \ldots, X_n \sim^{iid} NB(r, \theta)$, where NB$(r, \theta)$ refers to the negative binomial distribution. Assuming r is known, determine the maximum likelihood estimator of θ as a function of observation (realizations) $x_1, \ldots, x_n$ of $X_1, \ldots, X_n$.

(b) Calculate the Hessian of the log-likelihood function and verify that the estimator you found is indeed the unique maximizer of the log-likelihood function.

Solution
a) Letting $k = m - r$, we find the pmf for the $NH(r, \theta)$ to be $\binom{k+r+1}{k}(1-\theta)^k\theta^r$. Applying joint distributions, we find the likelihood function to be

$$l(\theta) = \prod_{i=1}^{n} \binom{k_i + r + 1}{k_i} (1-\theta)^{k_i}\theta^r$$

which we can take the log of to find the log-likelihood of:

$$l(\theta) = \sum_{i=1}^{n} (\log \binom{k_i + r + 1}{k} + k_i \log(1-\theta) + r \log \theta)$$

We must take the derivative with respect to θ of the likelihood function and set it to 0 to determine the $\hat{\theta}$.

$$\begin{aligned} \frac{dl}{d\theta} &= \sum_{i=1}^{n} (\frac{r}{\theta} - \frac{k_i}{1-\theta}) = 0 \\ &\Rightarrow \frac{nr}{\theta} = \sum_{i=1}^{n} \frac{k_i}{1-\theta} \\ &\Rightarrow \hat{\theta} = \frac{nr}{\sum_{i=1}^{n}(k_i) + nr} \end{aligned}$$

Notice that the binomal is not needed, as it is a constant.

b) To calculate the Hessian, we simply take the derivative with respect to θ again.

$$\begin{aligned}
\frac{dl}{d\theta} &= \sum_{i=1}^{n}(\frac{r}{\theta} - \frac{k_i}{1-\theta}) \\
\Rightarrow \quad \frac{d^2l}{d\theta^2} &= H = \sum_{i=1}^{n}(-\frac{r}{\theta^2} - \frac{k_i}{(1-\theta)^2}) \\
&= -\frac{nr}{\theta^2} - \sum_{i=1}^{n}\frac{k_i}{(1-\theta)^2} < 0
\end{aligned}$$

Since both terms: $-\frac{nr}{\theta^2}$ and $-\sum_{i=1}^{n}\frac{k_i}{(1-\theta)^2}$ are less than 0, we have determined the $\hat{\theta}$ calculated above is indeed a maximizer of the log-likelihood function.

2. Naive Bayes for Document Classification: Cars or Motorcycles?

In this problem, you will implement a Naive Bayes classifier on the Newsgroup20 dataset. The documents in the data set are messages sourced from 20 newsgroups (an online message forum). You will build a classifier to determine whether a message originated from a newsgroup about cars or motorcycles, suing the bag-of-words representation discussed in class. You should implement this algorithm yourself. Do not use existing implementations. The following section provides background for the Multinomial Naive Bayes model you will be implementing.

Naive Bayes Background
Each sample in the data set correspond to one document, and consists of a d-dimensional feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$ and a binary label $y \in \{0, 1\}$. Each feature x_j is a count of the number of times word j occurs in the document.

For a given feature vector $\mathbf{x}$, Naive Bayes makes a classification by estimating quantities in the following formula for the Bayes classifier:

$$\hat{y} = \arg\max_{k \in \{0,1\}} \pi_k \prod_{j=1}^{d} \Pr(X_j = x_j \mid Y = k).$$

Here, X_j is the jth feature, viewed as a random variable, Y is the label, and $\Pr(X_j = l \mid Y = k)$ is the probability that word j occurred l times in document i, given the document belongs to class k (bag of words model).
For multinomial Naive Bayes, we will model this class-conditional probability as

$$\Pr(X_j \; l \mid Y = k) = (p_{kj})^l$$

using the following estimate for p_{kj}:

$$\hat{p_{kj}} = \frac{n_{kj} + \alpha}{n_k + \alpha d}$$

where $n_k = \sum_{i:y_i=k} \sum_j x_{ij}$ is the total number of words in the label k documents, and $n_{kj} = \sum_{i:y_i=k} x_{ij}$ is the number of times word j appears in documents with label k, with smoothing parameter $\alpha = 1$ to ensure $\hat{p_{kj}} > 0$.

π_k denotes the class-k priors, which are estimated as $\hat{\pi_k} = \frac{m_k}{n}$, where $m_k = |\{i; y_i = k\}|$ is the number of label k documents.

Notice that in the classification rule, we take the product over probabilities, and we exponentiate a probability in the class-conditional estimate. In some situations, multiplying many numbers less than one can result in underflow, as the product becomes very small. The resulting numerical error can worsen performance of the classifier. Thankfully, underflow can be mitigated by doing operations in log-space, and converting back via exponentiation when needed.
In this problem you will implement the classifier

$$\hat{y} = \arg\max_{k \in \{0,1\}} \log \left(\hat{\pi_k} \prod_{j=1}^{d} (\hat{p_{kj}})^{x_j} \right) \tag{1}$$

The log does not change the maximizer since it is a strictly increasing function. Here and through out the course, logarithms are natural unless a different base is specified.

(a) Intuitively, p_{kj} is the probability that feature k appears once in a class-k document. What additional assumption about the data, in addition to the naive Bayes assumption, makes this intuition valid?

(b) By applying properties of the natural logarithm, simplify the expression on the right-hand side of (1), substituting the formula for $\hat{p_{kj}}$. Your final expression should not contain logs of products or divisors.

(c) Download the training and test datasets from Canvas under Files, Homework, HW2, hw2p2data.zip. hw2p2trainx.npy, hw2p2trainy.npy are the training features and labels respectively, hw2p2testx.npy and hw2p2testy.npy are the test data. The data can be loaded with numpy.load(filename). For example:
trainx = np.load("hw2p2trainx.npy")
trainy = np.load("hw2p2trainy.npy")
Here, trainx has dimensions $n_{train} \times d$, and trainy is an n_{train}-vector of 0s and 1s, denoting a message from the cars group or motorcycles group, respectively. The training set consists of $n_{train} = 1192$ samples, and the test set has $n_{test} = 794$ samples. There are $d = 1000$ features, where each feature denotes the number of times a particular word appeared in a document.
(i) Use the training data to estimate $\log p_{kj}$ for each class $k = 0, 1$ and for each feature

$j = 1, \ldots d$.
(ii) Also estimate the log-priors $\log \pi_k$ for each class.

(d) Use the estimates for $\log p_{kj}$ and $\log \pi_k$ to predict labels for the testing data. That is apply the decision rule with the samples in testx.npy and testy.npy. Report the test error.

(e) What would the test error be if we always predicted the same class, namely, the majority class from the training data? (Use this result as a sanity check for the error in part (d)).

(f) Submit all code used for (c), (d), and (e) as a single .py file. Submit this file to the corresponding assignment on Canvas. In addition, please also submit a .pdf version of your code (just print the .py file to pdf) and upload to gradescope for part (f). This will make is easier for graders who want to take a quick look at your code without running it. Your code should follow best coding practices including the use of comments, indentation, and descriptive variable names.

Solution

(a) The other assumption of the data that we need to make is to assume that the word occurrences follow a multinomial distribution. This can be thought of handling the 'j' part of p_{kj}.

(b) We define the Naive Bayes classifier to be

$$\hat{y} = \arg \max_{k \in \{0,1\}} \log(\hat{\pi}_k(\prod_{j=1}^{d}(\hat{p}_{kj})^{x_j}))$$

Using log properties we can rewrite this into a summation:

$$\begin{aligned} \hat{y} &= \arg \max_{k \in \{0,1\}} \log \hat{\pi}_k + \log(\prod_{j=1}^{d}(\hat{p}_{kj})^{x_j}) \\ &= \arg \max_{k \in \{0,1\}} \log \hat{\pi}_k + \sum_{j=1}^{d}(x_j \log(\hat{p}_{kj})) \end{aligned}$$

Using the definition of $\hat{p}_{kj}$, we can once again rewrite this, and simplify to remove any multiplication/division with logs, and arrive at the final answer:

$$\begin{aligned} \hat{y} &= \arg \max_{k \in \{0,1\}} \log \hat{\pi}_k + \sum_{j=1}^{d}(x_j \log(\frac{n_{kj} + \alpha}{n_k + \alpha d})) \\ \hat{y} &= \arg \max_{k \in \{0,1\}} \log \hat{\pi}_k + \sum_{i=1}^{d}(x_j(\log(n_{kj} + \alpha) - \log(n_k + \alpha d))) \end{aligned}$$

(c) (i) Estimated $\log p_{kj}$ for each class is attached as two seperate csv files on the Canvas Assignment.
(ii) Estimated $\log \hat{\pi}_0 = -0.6965$, $\log \hat{\pi}_1 = -0.6897$

(d) After predicting labels for the testing data, I found the test error to be 0.1259.

(e) When only predicting the majority class from the training data, I found the error to be 0.4987. This confirms that my error in the previous part is satisfactory, as it is $\sim 1/4$th the error when compared to only predicting the majority class from training data.

(f) The code for parts c, d, e is posted in the Index of this document. The .py file is found in the corresponding assignment on Canvas. The .pdf of the code is also uploaded to gradescope.

3. Logistic regression objective function

Consider the regularized logistic regression objective (penalized negative log-likelihood)

$$J(\theta) = -l(\theta) + \lambda\|\theta\|^2$$

where

$$l(\theta) = \sum_{i=1}^{n} \left[y_i \log\left(\frac{1}{1+e^{-\theta^T \tilde{x}_i}}\right) + (1-y_i)\log\left(\frac{e^{-\theta^T \tilde{x}_i}}{1+e^{-\theta^T \tilde{x}_i}}\right)\right]$$

Here $\theta = [b\ w_1 \dots\ w_d]^T$, $\tilde{x}_i = [1\ x_{i1} \dots\ x_{id}]^T$ and $y_i \in \{0, 1\}$.

(a) Show that if we change the label convention in logistic regression from $y \in \{0, 1\}$ to $y \in \{-1, 1\}$, then

$$-l(\theta) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \theta^T \tilde{x}_i)).$$

Introduce the notation $\phi(t) = \log(1+\exp(-t))$ so that, by the previous problem, the logistic regression regularized negative log-likelihood may be written

$$J(\theta) = \sum_{i=1}^{n} \phi(y_i \theta^T \tilde{x}_i) + \lambda\|\theta\|^2. \tag{2}$$

(b) Calculate the gradient of $J(\theta)$ using (2). Your answers may be in the form of a summation. Simplify your result so that it involves only the full vectors x_i (or $\tilde{x}_i$) and not their

components.

(c) Calculate the Hessian of $J(\theta)$. This will be a $(d+1) \times (d+1)$ matrix. Again your result may be in the form of a summation. Simplify your result so that it involves only the full vectors x_i (or $\tilde{x}_i$) and not their components.

(d) Use the previous result to argue that J is a convex function of θ when $\lambda \geq 0$, and strictly convex when $\lambda > 0$.

Solution

(a) When using the convention of $y \in \{0, 1\}$, we get:

$$l(\theta) = \sum_{i=1}^{n} \log \left(\frac{e^{-\theta^T \tilde{x}_i}}{1 + e^{-\theta^T \tilde{x}_i}} \right)$$

for the y_0 case, and

$$l(\theta) = \sum_{i=1}^{n} \log \left(\frac{1}{1 + e^{-\theta^T \tilde{x}_i}} \right)$$

for the y_1 case. Let us rewrite the given $l(\theta)$ for the $y \in \{-1, 1\}$ case:

$$l(\theta) = -\sum_{i=1}^{n} \log \left(1 + e^{y_i \theta^T \tilde{x}_i} \right) = \sum_{i=1}^{n} \log \left(\frac{1}{1 + e^{y_i \theta^T \tilde{x}_i}} \right)$$

We need to show that $y_1 = y_1$, and $y_{-1} = y_0$ for each convention's loss function. With y_1, we get:

$$l(\theta) = \sum_{i=1}^{n} \log \left(\frac{1}{1 + e^{-\theta^T \tilde{x}_i}} \right)$$

which is equal to the function for convention $y \in \{0, 1\}$. Letting y_{-1}, we get:

$$l(\theta) = \sum_{i=1}^{n} \log \left(\frac{1}{1 + e^{\theta^T \tilde{x}_i}} \right)$$

multiplying the numerator and denominator by $e^{-\theta^T \tilde{x}_i}$, we get

$$l(\theta) = \sum_{i=1}^{n} \log \left(\frac{e^{-\theta^T \tilde{x}_i}}{1 + e^{-\theta^T \tilde{x}_i}} \right)$$

which matches the y_0 for the $y \in \{0, 1\}$ convention. Thus we have shown the function give in the peroblem statement to be accurate for the convention $y \in \{-1, 1\}$.

(b) To find the $\nabla J(\theta)$ we do:

$$\begin{aligned}
\nabla J(\theta) &= \nabla(\sum_{i=1}^{n} \log\left(1 + e^{-y_i\theta^T\tilde{x}_i}\right) + \lambda\|\theta\|^2) \\
&= \nabla(\sum_{i=1}^{n} \log\left(1 + e^{-y_i\theta^T\tilde{x}_i}\right)) + 2\lambda\theta \\
&= \sum_{i=1}^{n}\left(\frac{1}{1 + e^{-y_i\theta^T\tilde{x}_i}}\nabla(1 + e^{-y_i\theta^T\tilde{x}_i})\right) + 2\lambda\theta \\
&= \sum_{i=1}^{n}\left(\frac{-y_i\tilde{x}_i e^{-y_i\theta^T\tilde{x}_i}}{1 + e^{-y_i\theta^T\tilde{x}_i}}\right) + 2\lambda\theta
\end{aligned}$$

We can simplify using the same trick from part (a) to get:

$$\nabla J(\theta) = \sum_{i=1}^{n}\left(\frac{-y_i\tilde{x}_i}{1 + e^{y_i\theta\tilde{x}_i}}\right) + 2\lambda\theta$$

(c) To find the Hessian, we simply take the gradient again:

$$\begin{aligned}
H &= \nabla(\sum_{i=1}^{n}\left(\frac{-y_i\tilde{x}_i}{1 + e^{y_i\theta^T\tilde{x}_i}}\right) + 2\lambda\theta) \\
&= 2\lambda I - y_i\tilde{x}_i\sum_{i=1}^{n}\nabla(1 + e^{y_i\theta^T\tilde{x}_i})^{-1} \\
&= 2\lambda I + y_i\tilde{x}_i\sum_{i=1}^{n}\frac{y_i\tilde{x}_i^T e^{y_i\theta^T\tilde{x}_i}}{(1 + e^{y_i\theta^T\tilde{x}_i})^2}
\end{aligned}$$

y_i^2 can be ignored as for each value of $y_i, y_i^2 = 1$. This gives us a final Hessian of:

$$H = \sum_{i=1}^{n}\left(\frac{e^{y_i\theta^T\tilde{x}_i}}{(1 + e^{y_i\theta^T\tilde{x}_i})^2}\tilde{x}_i\tilde{x}_i^T\right) + 2\lambda I$$

(d) For J to be convex/strictly convex, we need to show that H is PSD/PD. Since we know that xx^T is PD for all $x > 0$, we can conclude that the summation (first term) is also PD, as it is simply a weighted sum of a PD matrices, which preserves positive definteness. If $\lambda \geq 0$, we know that the term $2\lambda I$ is PSD, and thus the sum of a PSD matrix, and a PD matrix will give a PSD matrix, thus showing convexity. If we know that $\lambda > 0$, then the $2\lambda I$ term is PD, and thus the Hessian is also PD.

4. Logistic Regression for Fashion Classification

Download the files fashionmnistimages.npy, fashionmnistlabels.npy from Canvas under Files, Homework, HW2, hw2p24data.zip. This is a subset of the "Fashion MNIST" dataset, which contains images of different articles of clothing. This subset contains examples of coats and dresses.
The data file contains variables x and y, with the former containing images and ther latter labels. The images are stored as column vectors. To visualize an image, in Python run

```
import numpy as np
import matplotlib.pyplot as plt

x = np.load("fashion_mnist_images.npy")
y = np.load("fashion_mnisrt_labels.npy")
d, n- x.shape

i = 0 #Index of the image to be visualized
plt.imshow(np.reshape(x[:,i], (int(np.sqrt(d)), int(np.sqrt(d)))), cmap="Greys")
plt.show()
```

The above code can be found in the file loadfmnist.npy.

Newton's method finds a critical point of an objective function $J(\theta)$ by iterating

$$\theta_{t+1} = \theta_t - (\nabla^2 J(\theta_t))^{-1} \nabla J(\theta_t)$$

Implement Newton's method (a.k.a Newton-Raphson) to find a minimizer of the regularized negative log likelihood $J(\theta)$ from the previous problem. Set $\lambda = 1$. Use the first 5000 examples as training data, and the last 1000 as test data. As a termination criterion, let i be the final iteration as soon as $|J(\theta_i) - J(\theta_{i-1})|/J(\theta_{i-1}) \leq \epsilon := 10^{-6}$. Let the initial iterate θ_0 be the zero vector.

(a) Report the test error, the number of iterations run, and the value of the objective function after convergence.

(b) Generate a figure displaying 20 images in a 4×5 array. These images should be the 20 misclassified images for which the logistic regression classifier was least confident about its prediction (you will have to define a notion of confidence in a reasonable way - explain what this is). In the title of each subplot, indicate the true label of the image. What you should expect to see is some dresses that look kind of like coats and coats that look kind of like dresses.

(c) Submit your code as a single .py file to the corresponding assignment on Canvas designated for this purpose. In addition, please also submit a .pdf version of your code (just print the .py file to pdf) and upload to gradescope. This will make it easier for graders who want to take a quick look at your code without running it. Your code should follow best coding practices including the use of comments, indentation, and descriptive variable names.

Some additional remarks:

- Helpful Python commands and libraries: numpy.log, numpy.exp, numpy.sum, numpy.sign, numpy.zeros, numpy.repeat, str(), matplotlib.pyplot.
- Note that the labels in the data are ± 1, whereas the notes (at times) assume that the labels are 0 and 1.

Solution

(a) I am getting a test error of 0.088, a total of 4 iterations. and a final objective value of 0.4667. See index for code.

(b) From my output, we can check that the images received are very difficult to distinguish between dresses and coats:

(c) All code found in index of this document. The .py file is found in the corresponding assignment on Canvas. The .pdf of the code is also uploaded to gradescope.

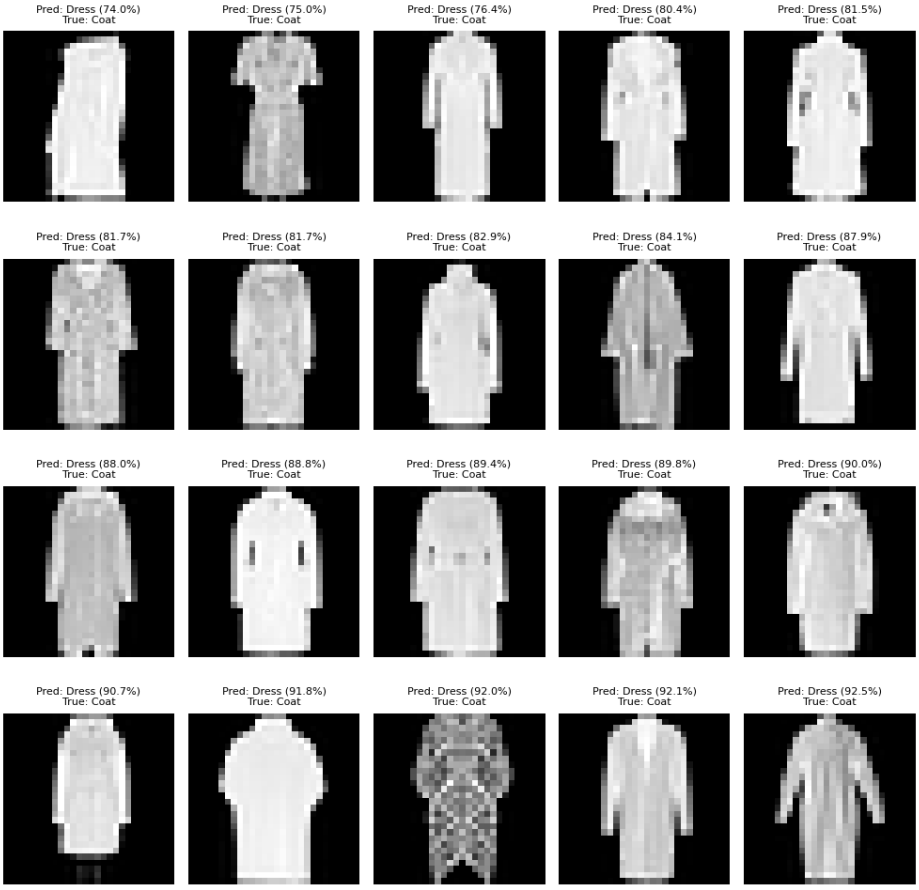

Figure 1: 4.2 Solution

Index

Question 2 Code:

```
import numpy as np
import matplotlib.pyplot as plt

#part c
train_x = np.load("hw2p2_train_x.npy") # dimensions: n_train x d (1192 x 1000)
train_y = np.load("hw2p2_train_y.npy")
# dimensions: n_train x 1 vector of labels (1192 x 1)

#n_train = 1192 samples
#n_test = 794 samples
#d = 1000 features (each feature denotes the number
#of times a word appeared in a document)

n_train, d = train_x.shape
```

```
alpha = 1

class_0_indices = (train_y == 0)
class_1_indices = (train_y == 1)

n_k0 = np.sum(train_x[class_0_indices]) # total words class 0
n_k1 = np.sum(train_x[class_1_indices]) # total words class 1

n_kj0 = np.sum(train_x[class_0_indices], axis=0)
n_kj1 = np.sum(train_x[class_1_indices], axis=0)

#p_kj = (n_kj + alpha) / (n_k + alpha*d) formula from derivation
p_kj0 = (n_kj0 + alpha) / (n_k0 + alpha*d)
p_kj1 = (n_kj1 + alpha) / (n_k1 + alpha*d)
log_p_kj0 = np.log(p_kj0)
log_p_kj1 = np.log(p_kj1)


# log(pi_hat_k) = log(m_k / n) formula from derivation
m_k0 = np.sum(class_0_indices)
m_k1 = np.sum(class_1_indices)
n = n_train

pi_hat_0 = m_k0 / n
pi_hat_1 = m_k1 / n
log_pi_hat_0 = np.log(pi_hat_0)
log_pi_hat_1 = np.log(pi_hat_1)

#(i) solution
np.savetxt("log_p_kj0.csv", log_p_kj0, delimiter=",")
np.savetxt("log_p_kj1.csv", log_p_kj1, delimiter=",")

#(ii) solution
print("Log priors:")
print(f"log pi_0: {log_pi_hat_0}, log pi_1: {log_pi_hat_1}")

# part d
test_x = np.load("hw2p2_test_x.npy")
test_y = np.load("hw2p2_test_y.npy")

#log probabilities for each class
log_prob_0 = log_pi_hat_0 + test_x @ log_p_kj0
log_prob_1 = log_pi_hat_1 + test_x @ log_p_kj1
```

```
predict = (log_prob_1 > log_prob_0).astype(int)

test_error = np.mean(predict != test_y)
print(f"Test error: {test_error}")

#part e, if always predict more majority class
majority_class = np.bincount(train_y).argmax()
baseline = np.full_like(test_y, majority_class)
baseline_error = np.mean(baseline != test_y)
print(f"Baseline error (when always predicting class
{majority_class}): {baseline_error}")
```

Question 4 Code:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.load("fashion_mnist_images.npy")
y = np.load("fashion_mnist_labels.npy").flatten()

lamda = 1
epsilon = 1e-6
max_iters = 1000
n = 5000

#train and test split
x_train, y_train = x[:, :n], y[:n]
y_train = y[:n]
x_test = x[:, n:]
y_test = y[n:]

theta = np.zeros((x_train.shape[0], 1))
m = x_train.shape[1]

#helper functions
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def J(theta):
    z = y_train * (theta.T @ x_train)
    loss = np.log(1 + np.exp(-z))
    return np.sum(loss)/n + lamda * np.linalg.norm(theta)**2

#newtons method
for iter in range(max_iters):
```

```
    y_pred = sigmoid(y_train.reshape(1,-1) * (theta.T @ x_train)).flatten()

    #gradient calculation
    grad = -(x_train @ ((1 - y_pred) * y_train).reshape(-1, 1)) / m + 2 * lamda * theta

    #hessian
    S = np.diag((y_pred * (1 - y_pred)).flatten())
    H = (x_train @ S @ x_train.T) / m + 2 * lamda * np.eye(x_train.shape[0])

    #newton method update
    theta -= np.linalg.inv(H) @ grad
    if(iter > 0 and abs(J_prev - J(theta)) / J_prev <= epsilon):
        break
    J_prev = J(theta)


#part a
y_test_pred = sigmoid(theta.T @ x_test).flatten()
y_test_pred_labels = (y_test_pred >= 0.5) * 2 - 1
test_error = np.mean(y_test_pred_labels != y_test)

print(f"Test error: {test_error}")
print(f"Iterations: {iter + 1}")
print(f"Final objective value: {J(theta)}")


#part b

#confidence scores
confidence = np.abs(0.5 - y_test_pred - 0.5) + 0.5

#find misclassed
misclassified = y_test_pred_labels != y_test
misclassified_indices = np.where(misclassified)[0]
misclassified_confidence = confidence[misclassified_indices]

#sort
sorted_indices = np.argsort(misclassified_confidence)
lowest_confidence_indices = misclassified_indices[sorted_indices[:20]]

# figure
plt.figure(figsize=(10, 10))
for i, idx in enumerate(lowest_confidence_indices):
    plt.subplot(4,5, i+1)
```

```
    img_size = int(np.sqrt(x_test.shape[0]))
    img = x_test[:, idx].reshape(img_size, img_size)
    plt.imshow(img, cmap='gray')
    confidence_percentage = confidence[idx] * 100
    pred_label = "Coat" if y_test_pred_labels[idx] == 1 else "Dress"
    true_label = "Coat" if y_test[idx] == 1 else "Dress"
    plt.title(f"Pred: {pred_label} ({confidence_percentage:.1f}%)\nTrue: {true_label}",
    fontsize=8)
    plt.axis('off')

plt.tight_layout()
plt.show()
```