1.(a)

$$L(\theta) = f(X_1, X_2, \dots X_n \mid \theta)$$

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

$$= \binom{n-1}{r+r} \theta^r \times (1-\theta)^{n-r}$$

$$\log L(\theta) = \log \binom{n-1}{n-r} + r \log(\theta) + (n-r-1) \log(1-\theta)$$

$$\frac{d}{d\theta} \log_e L(\theta) = \frac{r}{\theta} + \frac{n-r}{1-\theta} \times -1 = 0$$

$$\Rightarrow \quad \frac{r}{\theta} = \frac{n-r}{1-\theta}$$

$$\Rightarrow \quad r - r\theta = n\theta - r\theta$$

$$\Rightarrow \quad \frac{r}{n} = \theta \quad \ast\ast$$

1. (b)

The Hessian of $\log L(\theta)$

$$\frac{d^2}{d\theta^2}\log_e L(\theta) = \frac{-r}{\theta^2} + \frac{n-r}{(1-\theta)^2} \times -1$$

$$= \frac{-r(1-\theta)^2 - n\theta^2 + r\theta^2}{\theta^2(1-\theta)^2}$$

$$= \frac{-r(1-\theta)^2 + \theta^2(r-n)}{\theta^2(1-\theta)^2}$$

$-r(1-\theta)^2 < 0$, $\theta^2(r-n) < 0$, so the numerator < 0

$\theta^2(1-\theta)^2 > 0$

$\Rightarrow \nabla^2 \log L(\theta) < 0$

which means

$\log(L(\theta))$ has the unique maximize *

2. (a)

$X_1, \ldots, X_d$ need to be linear independant ※

, which means every feature won't influence others.

(b)

$$\hat{y} = \arg\max_{k \in \{0,1\}} \left[\log(\hat{\pi}_k) + \log\left(\prod_{j=1}^{d} (\hat{P}_{kj})^{x_j} \right) \right]$$

$$= \arg\max_{k \in \{0,1\}} \left[\log(\hat{\pi}_k) + \sum_{j=1}^{d} x_j \log(\hat{P}_{kj}) \right]$$ ※※

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

from $y \in \{0, 1\} \Rightarrow y \in \{-1, 1\}$

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

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

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

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

3. (b) $\nabla J(\theta) = \sum_{i=1}^{n} \phi'(y_i \theta^T \tilde{x}_i) + \lambda \|\theta\|^2)'$

$= \sum_{i=1}^{n} \frac{-y_i \tilde{x}_i}{1 + e^{-y_i \theta^T \tilde{x}_i}} \times e^{-y_i \theta^T \tilde{x}_i} + 2\lambda\theta$

$= \sum_{i=1}^{n} \frac{-e^{-y_i \theta^T \tilde{x}_i} \times y_i}{1 + e^{-y_i \theta^T \tilde{x}_i}} \tilde{x}_i + 2\lambda\theta$

$= \sum_{i=1}^{n} \frac{-y_i}{1 + e^{y_i \theta^T \tilde{x}_i}} \tilde{x}_i + 2\lambda\theta$ ※

(c) $\nabla^2 J(\theta)$

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

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

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

3.(d)

$$\nabla^2 J(\theta) = \sum_{i=1}^{n} \frac{x_i^2 e^{x_i \theta^T \tilde{x}_i}}{(1+e^{x_i \theta^T \tilde{x}_i})^2} \tilde{x}_i^2 + 2\lambda I$$

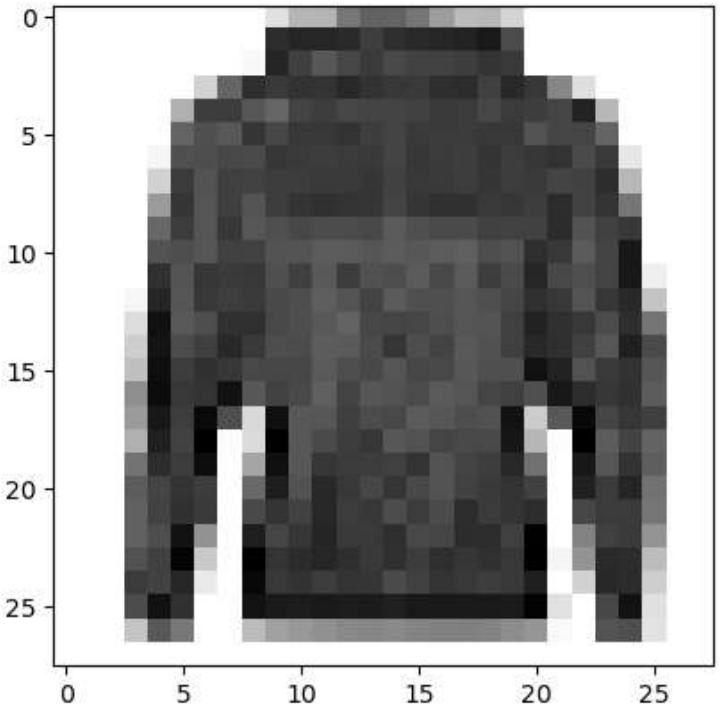
always >0 (numerator); always >0 (denominator); always >0 ($\tilde{x}_i^2$); >0 ($2\lambda I$)

when $\lambda \geq 0$ $J(\theta)$ has a local minimum

$\Rightarrow$ $J(\theta)$ is a convex function

$\lambda > 0$ $\Rightarrow$ $J(\theta)$ is strictly convex ※

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
x = np.load("fashion_mnist_images.npy")
y = np.load("fashion_mnist_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()
```



In [2]:
```
x = np.insert(x, 0, 1, axis = 0)
train_x = x[:, :5000]
test_x = x[:, 5000:]
train_y = y[:, :5000]
test_y = y[:, 5000:]
#print(x.shape[0])
```

In [3]:
```
# Calculate negative log-likelihood
def J(theta, X, y, lambda_val):
    m = len(y.T)
    l = 0
    for i in range(0, m):
        x_i = np.array(X[:, i]).reshape((len(X[:, i]), 1)) # d
        l += np.log(1 + np.exp(-y[0][i] * np.dot(np.transpose(theta), x_i)))
    J = l + lambda_val * np.sum(theta**2)
    return J

# Calculate the gradient and Hessian of the function
def gradient_and_hessian(theta, X, y, lambda_val):
    I = np.diag(np.ones(X.shape[0]), 0)
    m = len(y.T)
    gradient = np.zeros((X.shape[0], 1)) + 2 * lambda_val * theta
    hessian = np.zeros((X.shape[0], X.shape[0])) + 2 * lambda_val * I #don't forget I
    # calculate gradient
    # calculate hessian
    for i in range(0, m):
        x_i = np.array(X[:, i]).reshape((len(X[:, i]), 1)) # d
        gradient += -y[0][i] / (1 + np.exp(y[0][i] * np.dot(np.transpose(theta), x_i))) * x_
        sigmoid = (1 + np.exp(y[0][i] * np.dot(np.transpose(theta), x_i))) ** (-2)
        hessian += sigmoid * np.exp(y[0][i] * np.dot(np.transpose(theta), x_i)) * np.dot(x_i
        #print(gradient)

    return gradient, hessian
```

In [4]:
```
theta = np.zeros((x.shape[0], 1))
lambda_val = 1
epsilon = 1e-6
#print(theta.shape)
```

In [5]:
```
iterations = 0
pre_J = J(theta, train_x, train_y, lambda_val)
while True:
    #print("start")
    gradient, hessian = gradient_and_hessian(theta, train_x, train_y, lambda_val)
    #print(theta)
    theta -= np.linalg.inv(hessian).dot(gradient)
    #print(theta)
    current_J = J(theta, train_x, train_y, lambda_val)
    iterations += 1
    print(iterations)
    print((current_J - pre_J) / pre_J)
    if np.abs(current_J - pre_J) / pre_J <= epsilon:
        break
    pre_J = current_J
```

```
1
[[-0.67356582]]
2
[[-0.37893995]]
3
[[-0.2312517]]
4
[[-0.11356009]]
5
[[-0.03905217]]
6
[[-0.00715491]]
7
[[-0.0003817]]
8
[[-1.58437748e-06]]
9
[[-3.41397388e-11]]
```

In [84]:
```
def predict(theta, X):
    exp = np.exp(-X.T.dot(theta))
    raw_predictions = 1 / (1 + exp)
    predictions = np.zeros((raw_predictions.shape[0], 1))
    #print(predictions)
    for i in range(0, raw_predictions.shape[0]):
        if raw_predictions[i] >= 0.5:
            predictions[i] = 1
        else:
            predictions[i] = -1
    predictions = np.array(predictions.reshape(1, 1000))
    #print(predictions)
    return predictions, raw_predictions

test_predictions, raw_predictions = predict(theta, test_x)
#print(test_predictions != test_y)
test_error = np.mean(test_predictions != test_y)
```

In [85]:
```
print(f"Test_error: {test_error}\nIterations: {iterations}\nObjective Function: {current_J}"
```

```
Test_error: 0.034
Iterations: 9
Objective Function: [[456.63896506]]
```

In [94]:
```
predictions = raw_predictions
true_labels = test_y

# Calculate the confidence scores
confidence_scores = np.abs(predictions - 0.5)  # I define the less confidence one if it is ne
                                               # model cannot actually tell which label it sh
confidence_scores = np.array(confidence_scores.reshape(1, -1))
A = np.argsort(confidence_scores)
# Select the top 20 misclassified samples with the lowest confidence scores
least20_confident_indices = A[0][:20]
#print(least20_confident_indices)
# Create a 4x5 grid for plotting
fig, axes = plt.subplots(4, 5, figsize=(12, 10))
fig.subplots_adjust(hspace=0.5)

# Plot the selected misclassified images
for i, ax in enumerate(axes.ravel()):
    index = least20_confident_indices[i]
    image = test_x[:, index] # Replace with your actual test data shape
    true_label = true_labels[0][index]
    image = np.delete(image, 0)
    d = test_x[:, index].shape
    ax.imshow(np.reshape(image, (int(np.sqrt(d)),int(np.sqrt(d)))), cmap='gray')
    ax.set_title(f"True Label: {true_label}")
    ax.axis('off')

plt.show()
```


True Label: 1.0
True Label: 1.0
True Label: 1.0
True Label: 1.0
True Label: -1.0
True Label: 1.0
True Label: 1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0
True Label: 1.0
True Label: 1.0
True Label: 1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0
True Label: -1.0

In [ ]:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
train_x = np.load("hw2p2_data/hw2p2_train_x.npy")
train_y = np.load("hw2p2_data/hw2p2_train_y.npy")
test_x = np.load("hw2p2_data/hw2p2_test_x.npy")
test_y = np.load("hw2p2_data/hw2p2_test_y.npy")
train_x.shape[1]
```

Out[2]:
```
1000
```

In [3]:
```
def probability(x, y, z):
    nk = 0
    nkj = np.zeros(x.shape[1])
    for i in range(0, x.shape[0]):
        if y[i] == z:
            nk += 1
    # need to work on this
    for j in range(0, x.shape[1]):
        for i in range(0, x.shape[0]):
            if (y[i] == z):
                nkj[j] += x[i][j]

    #print(nkj)
    p = (nkj + 1) / (nk + x.shape[0])
    return p, nk
def prediction(x, p, logpi):
    y = np.zeros(x.shape[0])
    for i in range(0, x.shape[0]):
        sum_1 = 0
        sum_0 = 0
        for j in range(0, x.shape[1]):
            if(x[i][j] == 1):
                sum_1 += p[1][j]
                sum_0 += p[0][j]

        #print(sum_1 + logpi[1], sum_0 + logpi[0])
        if(sum_1 + logpi[1] > sum_0 + logpi[0]):
            y[i] = 1
        elif(sum_1 + logpi[1] < sum_0 + logpi[0]):
            y[i] = 0
    return y
```

In [4]:
```
p_1, n_1 = probability(train_x, train_y, 1) #probability
p_0, n_0 = probability(train_x, train_y, 0)
pi_k1 = n_1 / train_x.shape[0]
pi_k0 = n_0 / train_x.shape[0]
```

In [5]:
```
logp_1 = np.log(p_1)
logp_0 = np.log(p_0)
logpi_k1 = np.log(pi_k1)
logpi_k0 = np.log(pi_k0)
logp = [logp_0, logp_1]
logpi_k = [logpi_k0, logpi_k1]
```

In [10]:
```
print(f"For k = 1\n log(p): {logp_1}, log(pi): {logpi_k1}")
print(f"For k = 0\n log(p): {logp_0}, log(pi): {logpi_k0}")
```

```
For k = 1
 log(p): [-5.18738581 -5.88053299 -5.41052936 -5.88053299 -5.54406075 -7.4
899709
 -7.4899709  -4.85091357 -7.4899709  -7.4899709  -7.4899709  -6.10367654
 -5.88053299 -7.4899709  -6.79682372 -7.4899709  -7.4899709  -5.88053299
 -7.4899709  -5.54406075 -5.29274632 -7.4899709  -7.4899709  -7.4899709
 -7.4899709  -7.4899709  -5.41052936 -7.4899709  -7.4899709  -7.4899709
 -7.4899709  -7.4899709  -7.4899709  -7.4899709  -7.4899709  -7.4899709
 -7.4899709  -7.4899709  -7.4899709  -5.88053299 -7.4899709  -7.4899709
 -7.4899709  -7.4899709  -7.4899709  -7.4899709  -5.41052936 -7.4899709
 -6.79682372 -6.79682372 -5.69821143 -5.69821143 -7.4899709  -7.4899709
 -7.4899709  -7.4899709  -7.4899709  -6.39135861 -7.4899709  -7.4899709
 -4.59959914 -5.88053299 -5.54406075 -5.41052936 -5.88053299 -5.69821143
 -6.10367654 -5.00506425 -5.29274632 -5.88053299 -7.4899709  -7.4899709
 -7.4899709  -6.79682372 -7.4899709  -7.4899709  -5.41052936 -6.39135861
 -6.79682372 -7.4899709  -7.4899709  -7.4899709  -7.4899709  -5.41052936
 -6.39135861 -7.4899709  -5.00506425 -7.4899709  -6.79682372 -4.05598369
 -5.41052936 -4.7819207  -7.4899709  -7.4899709  -5.54406075 -6.79682372
 -7.4899709  -7.4899709  -6.79682372 -3.04731964 -7.4899709  -4.44544846
```

In [6]:
```
y = prediction(test_x, logp, logpi_k)
#print(y)
```

In [8]:
```
test_error = (y - test_y) ** 2
print(f"test error: {np.sum(test_error)}")
```

```
test error: 134.0
```

2.(e)

if we always predict the same class, the test error will be about 0.5

In [ ]:
```

```