1) I certify that all solutions are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted.

X ______________________

People I collaborated with: Random office hours students, Madeline (SID: 3036751976)

2) 1) $\frac{\partial J(w)}{\partial s} =$

$\frac{\partial s}{\partial \gamma} = s(\gamma)(1 - s(\gamma))$

$$\boxed{\nabla_w J = -X^T (y - s(Xw)) \text{ when } s(Xw) = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix}}$$

Taken from lec 10 page 4 from notes.

2) $\nabla_w J = \cancel{-X^T y} - X^T s(Xw)$

$$\boxed{\nabla_w^2 J = X^T \Omega X}$$

3) $v - (\nabla^2 J(w))^{-1} \nabla J(v)$

$$\boxed{v - X s(Xw)^{-1} \cdot - X^T \Omega X}$$

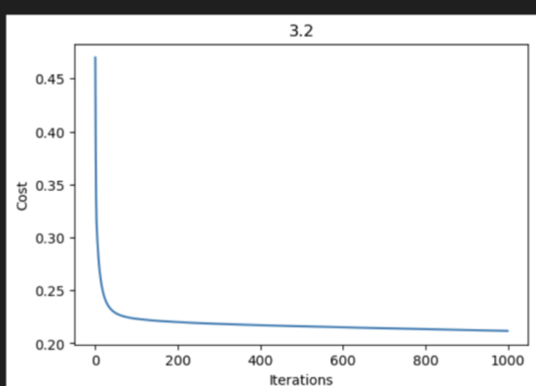
4)
$$\boxed{\begin{aligned} s^{(0)} &= [0.9479, 0.8808, 0.8022, 0.5250] \\ w^{(1)} &= [1.3247, 3.0499, -6.8291] \\ s^{(1)} &= [0.9474, 0.9746, 0.0312, 0.1044] \\ w^{(2)} &= [1.3660, 4.1575, -9.1996] \end{aligned}}$$

3) 1) $\boxed{\frac{1}{m}(h(X)-y)X}$

2)

```python
#Question 3.2
def gradient_descent(data_matrix, target_vector, regularization, learning_rate, iterations):
    num_samples, num_features = data_matrix.shape
    zero = np.zeros(num_features) #weights matrix
    costs = np.zeros(iterations)

    for i in range(0,iterations):
        hypothesis = axpit(data_matrix.dot(zero))
        gradient = (1 / num_samples) * data_matrix.T.dot(hypothesis - target_vector) + (regularization/len(data_matrix)) * np.hstack(([0], zero[1:]))
        zero -= learning_rate * gradient
        costs[i] = (-1/len(data_matrix)) * (target_vector.dot(np.log(axpit(data_matrix.dot(zero)))) + (1 - target_vector).dot(np.log(1 - axpit(data_matrix.dot(zero)))))

    return costs

data = scipy.io.loadmat('data.mat')

#print(data.keys())

x_data = data['X']
y_data = data['y']
y_data = y_data.flatten()

mean = np.mean(x_data)
std_deviation = np.std(x_data)

x_norm = (x_data - mean) / std_deviation
x_norm = np.hstack((x_norm, np.ones((x_norm.shape[0], 1))))

costs = gradient_descent(x_norm, y_data, 0.1, 1, 1000)
print(costs[costs.size - 1])

plt.figure(figsize=(6, 4))
plt.plot(range(1000), costs)
plt.title('3.2')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

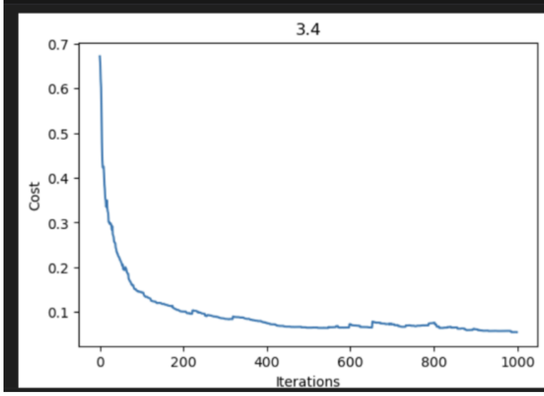
```
0.2117175807296077
```



3) $\boxed{(h(x) - y)x}$

4)

```python
#Question 3.4
def gradient_descent_stochastic(data_matrix, target_vector, learn, regularization, iterations):
    num_samples, num_features = data_matrix.shape
    zero = np.zeros(num_features)
    costs = np.zeros(iterations)

    for i in range(iterations):
        sample_idx = np.random.randint(num_samples)
        X_i = data_matrix[sample_idx, :].reshape(1, -1)
        y_i = target_vector[sample_idx].reshape(-1)

        hypothesis = axpit(X_i.dot(zero))
        gradient = X_i.T.dot(hypothesis - y_i) + regularization * np.hstack(([0], zero[1:]))
        zero -= learn * gradient

        costs[i] = (-1/len(data_matrix)) * (target_vector.dot(np.log(axpit(data_matrix.dot(zero)))) + (1 - target_vector).dot(np.log(1 - axpit(data_matrix.dot(zero)))))

    return costs


dataMat = scipy.io.loadmat('data.mat')

x_data = data['X']
y_data = data['y']
y_data = y_data.flatten()

mean = np.mean(x_data, axis = 0)
std_deviation = np.std(x_data, axis = 0)

x_norm = (x_data - mean) / std_deviation

x_norm = np.hstack((np.ones((x_norm.shape[0], 1)), x_norm))

costs = gradient_descent_stochastic(x_norm, y_data, 0.1, 0.01, 1000)
#print(costs[costs.size - 1])

plt.figure(figsize=(6, 4))
plt.plot(range(1000), costs)
plt.title('3.4')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

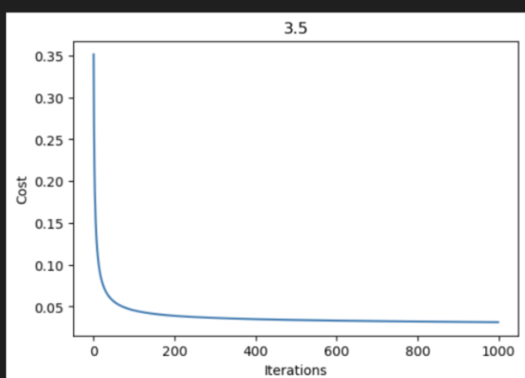


5)

```python
#Question 3.5
def cost(x, y, weights, regularization):
    len_y = len(y)
    sigma = axpit(x.dot(weights))
    return (-1/len_y) * (y.dot(np.log(sigma)) + (1 - y).dot(np.log(1 - sigma)))


def step_grad(data_matrix, target_vector, regularization, learning_rate, iterations):
    num_samples, num_features = data_matrix.shape
    zero = np.zeros(num_features) #weights matrix
    costs = np.zeros(iterations)

    for i in range(0,iterations):
        hypothesis = axpit(data_matrix.dot(zero))
        gradient = (1 / num_samples) * data_matrix.T.dot(hypothesis - target_vector) + (regularization/len(data_matrix)) * np.hstack(([0], zero[1:]))
        zero -= learning_rate * gradient
        costs[i] = (-1/len(data_matrix)) * (target_vector.dot(np.log(axpit(data_matrix.dot(zero)))) + (1 - target_vector).dot(np.log(1 - axpit(data_matrix.dot(zero)))))

    return costs


data = scipy.io.loadmat('data.mat')

x_data = data['X']
y_data = data['y']
y_data = y_data.flatten()

mean = np.mean(x_data, axis = 0)
std_deviation = np.std(x_data, axis = 0)

x_norm = (x_data - mean) / std_deviation

x_norm = np.hstack((np.ones((x_norm.shape[0], 1)), x_norm))

costs = step_grad(x_norm, y_data, 1, 1, 1000)
print(costs[costs.size - 1])

plt.figure(figsize=(6, 4))
plt.plot(range(1000), costs)
plt.title('3.5')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```

✓ 1.3s

```
0.03134023293037328
```



6) Kaggle Username: Christopher Avakian
Kaggle Score: 0.993

All I did was use my 3.2 solution, it seemed to give the lowest cost. I just adjusted numbers such as number of iterations, and the regularization parameter and learning rate, until I got a lower number.

```python
#Question 3.6
def cost(x, y, weights, regularization):
    len_y = len(y)
    sigma = axpit(x.dot(weights))
    return (-1/len_y) * (y.dot(np.log(sigma)) + (1 - y).dot(np.log(1 - sigma)))


def gradient_descent(data_matrix, target_vector, regularization, learning_rate, iterations):
    num_samples, num_features = data_matrix.shape
    zero = np.zeros(num_features) #weights matrix
    costs = np.zeros(iterations)

    for i in range(0,iterations):
        hypothesis = axpit(data_matrix.dot(zero))
        gradient = (1 / num_samples) * data_matrix.T.dot(hypothesis - target_vector) + (regularization/len(data_matrix)) * np.hstack(([0], zero[1:]))
        zero -= learning_rate * gradient
        costs[i] = (-1/len(data_matrix)) * (target_vector.dot(np.log(axpit(data_matrix.dot(zero)))) + (1 - target_vector).dot(np.log(1 - axpit(data_matrix.dot(zero)))))

    return costs, zero


dataMat = scipy.io.loadmat('data.mat')

x_data = data['X']
y_data = data['y']
y_data = y_data
y_data = y_data.flatten()

mean = np.mean(x_data, axis = 0)
std_deviation = np.std(x_data, axis = 0)

x_norm = (x_data - mean) / std_deviation

x_norm = np.hstack((np.ones((x_norm.shape[0], 1)), x_norm))

x_training_set = x_norm
y_training_set = y_data

costs, weights = gradient_descent(x_training_set, y_training_set, 0.1, 1, 5000)

test_data = (data['X_test'] - mean) / std_deviation
test_data = np.hstack((np.ones((test_data.shape[0], 1)), test_data))
probabilities = axpit(test_data.dot(weights))
predictions = (probabilities >= 0.5).astype(int)

results_to_csv(predictions)
```

4) 1) $f(w \mid X, y) = f(y \mid X, y) = \dfrac{f(y \mid X, w) \times \text{prior } f(w')}{f(y \mid X)}$

$$\boxed{\prod_{i=1}^{n} \frac{\frac{1}{\sigma\sqrt{2\pi}} \exp\left[-(y - w \cdot x)^2 / (2\sigma^2)\right] \cdot e^{-|w|/b} / 2b}{f(y \mid X)}}$$

2) $$\sum_{i=1}^{n} \frac{\cancel{\ln \frac{1}{\sigma\sqrt{2\pi}}} \left(-(y - w \cdot x)^2 / \cancel{(2\sigma^2)}\right) \cdot \frac{-|w|/b}{\ln(2b)}}{\ln(\cancel{f(y \mid X)})}$$

$$\boxed{\lambda = \frac{-1}{\ln(2b)\, b}}$$

therefore simplifying gives us

$$\sum_{i=1}^{n} (y_i - w \cdot x_i)^2 - \lambda \|w\|_1$$

5) 1) Code

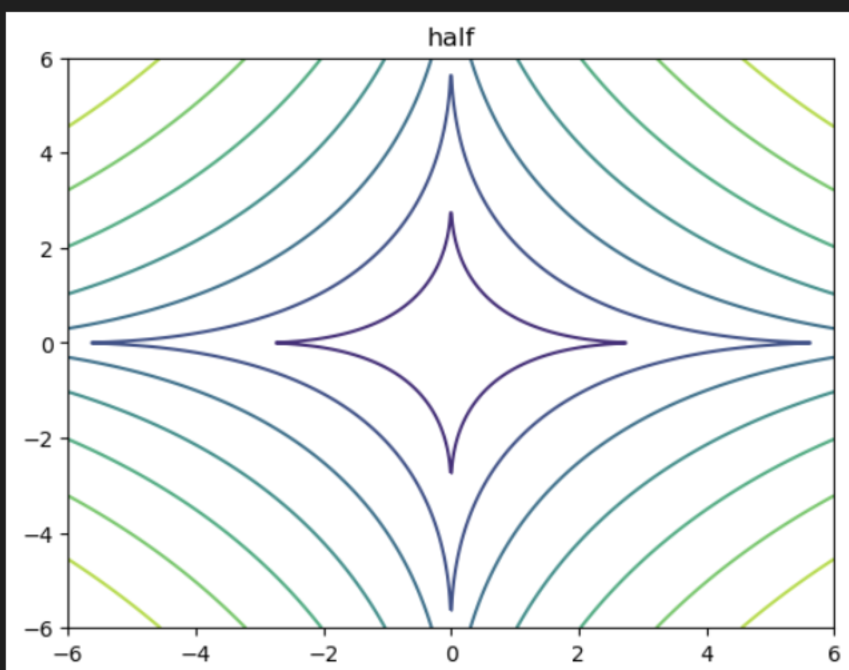
```
#Question 5.1
X, Y = np.meshgrid(np.linspace(-6, 6, 1000), np.linspace(-6, 6, 1000))

half = (np.abs(X)**0.5 + np.abs(Y)**0.5)**2

one = np.abs(X) + np.abs(Y)

two = np.sqrt(X**2 + Y**2)

plt.show()
```

[51]

```
plt.contour(X, Y, half)
plt.title("half")
```

[52]

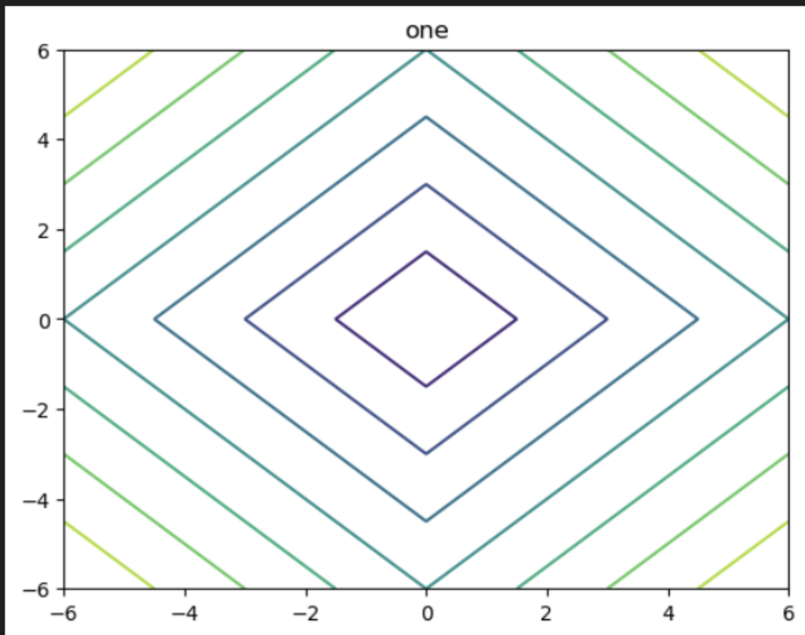
```
Text(0.5, 1.0, 'half')
```



```
plt.contour(X, Y, one)
plt.title("one")
```

[53]

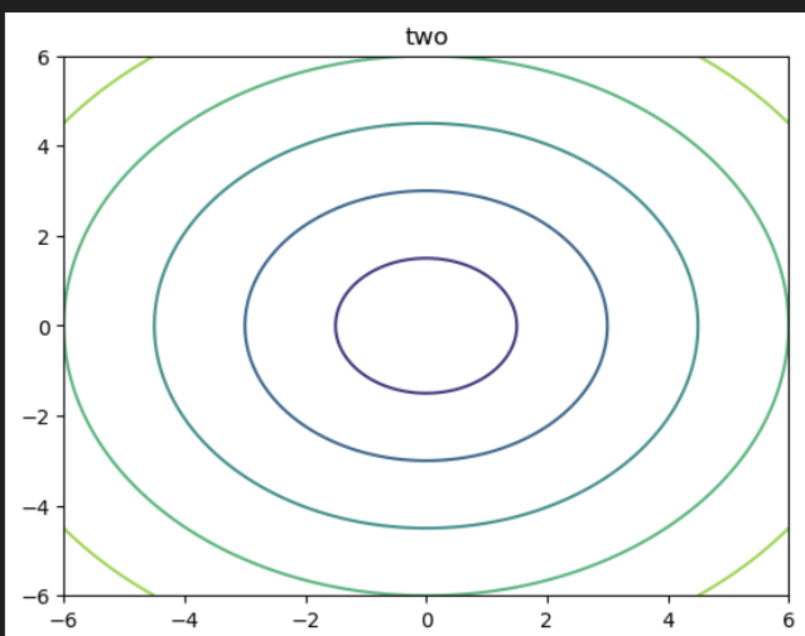
```
Text(0.5, 1.0, 'one')
```




```
plt.contour(X, Y, two)
plt.title("two")
```

[54]

```
Text(0.5, 1.0, 'two')
```



2) $= (X\omega - y)^T (X\omega - y)$

$= (\omega^T X^T - y^T)(X\omega - y)$

$$\omega^T \cancel{X^T X} \omega - \omega^T X^T y - y^T X\omega + y^T y$$

$$\omega^T \omega - \omega^T X^T y - y^T X\omega + y^T y$$

$$\lambda \|w\|_1 = \lambda \sum |w_i|$$

$$\underbrace{\omega^T \omega - \omega^T X^T y - y^T X\omega}_{\text{Express as Scalar summation form}} + \underbrace{y^T y}_{\|y\|^2} + \lambda \sum_{i=1}^{d} |w_i|$$

$$\|y\|^2 + \sum_{i=1}^{d} \lambda |\omega_i| + \omega^T \omega - \omega^T X^T y - y^T X\omega$$

$$f(x, \omega) = \lambda |\omega_i| + \omega^T \omega - \omega^T X^T y - y^T X\omega$$

$$\boxed{\|y\|^2 + \sum_{i=1}^{d} f(x, \omega)}$$

3) $\omega^i > 0$:

$$\nabla f(x_i^*, \omega_i) + \lambda = 0$$

$\omega^i < 0$:

$$\nabla f(x_i^*, \omega_i) - t = 0$$

$\omega^i = 0$:

If the other two properties don't hold

4) $$\nabla J_2(\omega) = 2\alpha^T(X\omega^\# - y) + 2\lambda\omega^\#$$

$$0 = 2\alpha^T(X\omega^\# - y) + 2\lambda\omega^\#$$

$$0 = 2X^T X\omega^\# - 2X^T y + 2\lambda\omega^\#$$

$$2X^T y = 2\omega^\# + 2\lambda\omega^\#$$

$$2X^T y = \omega^\#(2 + 2\lambda)$$

$$\boxed{\omega^\# = \frac{2x^T y}{2 + 2\lambda}}$$

5) ω^* more likely to be sparse since the $+\lambda\|\omega\|_1$ means that it can turn the ω value to 0, while the $\|\omega\|_2$ means so that there is a square, terms can get closer to 0, but won't reach 0.

References:

- https://en.wikipedia.org/wiki/Gradient
- https://en.wikipedia.org/wiki/Whitening_transformation
- https://en.wikipedia.org/wiki/Sigmoid_function#:~:text=A%20sigmoid%20function%20is%20a%20bounded%2C%20differentiable%2C%20real,at%20each%20point%20and%20exactly%20one%20inflection%20point.
- https://en.wikipedia.org/wiki/Gradient_descent
- https://en.wikipedia.org/wiki/Stochastic_gradient_descent
- https://stackoverflow.com/questions/43431236/how-to-perform-an-unregularized-logistic-regression-using-scikit-learn
- https://www.analyticsvidhya.com/blog/2021/08/performance-comparision-of-regularized-and-unregularized-regression-models/
- https://web.stanford.edu/~jurafsky/slp3/5.pdf