**ECE 50024: Homework 3**

**Manish Kumar Krishne Gowda, 0033682812 (Spring 2023)**

**Exercise 1:**

(i)

In logistic regression, we aim to find a hyperplane that separates the two classes of data. This hyperplane can be represented by the equation:

w^T x + w\_0= 0

The output of the logistic regression model as:

y = sigmoid(w^T x + w\_0)

If the output y is greater than or equal to 0.5, we predict that x belongs to the positive class, otherwise we predict that x belongs to the negative class.

if the two classes of data are linearly separable, it means that there exists a hyperplane that perfectly separates the two classes without any misclassifications. In other words, there exists a weight vector w and an intercept w\_0 such that:

w^T x\_i + w\_0 > 0 if x\_i belongs to Class1

w^T x\_i + w\_0 < 0 if x\_i belongs to Class0

This is equivalent to

y\_i = sigmoid(w^T x\_i + w\_0) > 0.5 if x\_i belongs to Class1

y\_i = sigmoid(w^T x\_i + w\_0) < 0.5 if x\_i belongs to Class0

Since the classes are perfectly separable, we can choose the weight vector w and the intercept w\_0 such that all the above conditions hold true. when we try to maximize the likelihood function of logistic regression to find the optimal values of w and w\_0. The likelihood function can be written as:

Text, letter

Description automatically generated

Which is equivalent to

Text

Description automatically generated with low confidence

When the classes are perfectly separable, it turns out that there exists no global minimum of the negative log-likelihood function. This is because we can always find a solution with a lower value of the negative log-likelihood function by increasing the magnitude of the weight vector w and the intercept w\_0. As we increase the magnitude of w, the hyperplane becomes steeper and steeper, which means that the distance between the positive and negative examples becomes larger and larger. As we increase the magnitude of w\_0, the hyperplane shifts farther away from the origin. As we keep increasing the magnitude of w and w\_0, we can always find a solution that perfectly separates the two classes of data with a lower value of the negative log-likelihood function. This is because the sigmoid function saturates as its input approaches infinity, which means that increasing the magnitude of w and w\_0 beyond a certain point does not affect the output of the sigmoid function for any input.

We also have

𝑦⋅𝑤^T𝑥≥0  for every data point (𝑥,𝑦).

We can rewrite this as 𝑦||𝑤||2||𝑥||2 cos𝜃≥0 where 𝜃 is the angle between the vectors 𝑥 and 𝑤. Therefore to minimize J(𝜃), we want the exponent to be as negative as possible (i.e. we want 𝑦||𝑤||2||𝑥||2 cos𝜃 as large and positive as possible). By separability of the data we know there is some vector 𝑤 such that the exponent is positive for every data point. Thus ||𝑤||2 is increased without bound.

ii)

If we restrict ∥w∥2 ≤ c1 and |w0| < c2 for some c1, c2 > 0, then we are adding a constraint on the magnitude of the weight vector w and the intercept w0. This means that we are limiting the "complexity" of the logistic regression model. In this case, the optimization problem for logistic regression becomes a constrained optimization problem, where we need to find the values of w and w0 that minimize the negative log-likelihood function subject to the constraints:

∥w∥2 ≤ c1

|w0| < c2

When we add constraints on the magnitude of w and w0, it can affect the performance of the logistic regression model. If the classes are perfectly separable, then it is possible that the constraint limits the ability of the model to find the optimal solution. This is because the optimal solution may require a larger magnitude of the weight vector and/or the intercept than the constraints allow. In this case, the model may not be able to achieve zero training error, even though the classes are perfectly separable.

On the other hand, if the classes are not perfectly separable, then the constraint can help prevent overfitting by limiting the complexity of the model. In this case, the model may not be able to achieve zero training error, but it may perform better on unseen data.

In general, the choice of the values of c1 and c2 would depend on the specific problem and the available data. If we have a small amount of data, it may be better to use a larger value of c1 and a smaller value of c2 to avoid overfitting. If we have a large amount of data, we may be able to use a smaller value of c1 and a larger value of c2 without overfitting.

If the logistic regression algorithm is not converging, there are several approaches that can be taken to counter the issue:

1. **Regularization**: Regularization can help prevent overfitting and improve convergence. L1 or L2 regularization can be used to add a penalty term to the loss function, which encourages the model to have smaller weights. This can lead to a simpler model and faster convergence.
2. **Different optimization algorithm**: The standard optimization algorithm used in logistic regression is gradient descent, but sometimes this algorithm may not converge. In such cases, other optimization algorithms such as stochastic gradient descent (SGD), mini-batch SGD, or Newton's method can be used.
3. **Initialization**: The initial values of the weight vector and intercept can affect the convergence of the logistic regression algorithm. Sometimes, starting with different initial values can help improve convergence.
4. **Adding more training data**: If the algorithm is not converging, adding more training data can help by reducing the effect of noise and outliers in the data.
5. Using different loss functions: Logistic regression uses the negative log-likelihood loss function. However, in some cases, using a different loss function such as hinge loss or squared loss may help improve convergence.

iii)

Linear separability of data may or may not cause convergence for the other linear classifiers. While the concept of linear separability is related to linear classifiers, it is not the only factor that determines the convergence of a linear classifier. Convergence depends on various factors such as the optimization algorithm, learning rate, initialization, regularization, and the complexity of the model.

For example, support vector machines (SVMs) are also linear classifiers that can handle linearly separable data. SVMs use a different loss function and optimization algorithm compared to logistic regression. Even when the data is linearly separable, it is possible for SVMs to have non-convergence issues if the hyperparameters such as the regularization parameter or kernel choice are not chosen appropriately.

**Exercise 2:**

(a)

Text, letter

Description automatically generated

(b), (c) and (d)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93 | **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  class0 = []  class1 = []  # Reading csv file for male data  **with** open("data/homework4\_class0.txt", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=' ')  **for** row **in** reader:  row = [i **for** i **in** row **if** i != '']  class0.append(list(np.float\_(row)))  class0 = np.array(class0)  csv\_file.close()  **print**(class0.shape)  **with** open("data/homework4\_class1.txt", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=' ')  **for** row **in** reader:  row = [i **for** i **in** row **if** i != '']  class1.append(list(np.float\_(row)))  class1 = np.array(class1)  csv\_file.close()  **print**(class1.shape)  #least squares  N = class0.shape[**0**] + class1.shape[**0**]  d = **3**  x = np.vstack((class0[:,**0**:**2**],class1[:,**0**:**2**]))  X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2  y = np.vstack((np.zeros((class0.shape[**0**],**1**)),np.ones((class1.shape[**0**],**1**))))  lambd = **0.0001**  theta = cp.Variable((d,**1**))  log\_likelihood = cp.sum(cp.multiply(y, X @ theta) - cp.logistic(X @ theta))  prob = cp.Problem(cp.Maximize(log\_likelihood/N - lambd \* cp.norm(theta, **2**)))  prob.solve()  theta\_cap\_cvx = theta.value  **print**(f"theta\_cap by cvx method :**\n** {theta\_cap\_cvx}")  plt.scatter(class0[:,**0**], class0[:,**1**], edgecolor ="blue", marker ="o")  plt.scatter(class1[:,**0**], class1[:,**1**], c ="red", marker =".")  line\_x = np.linspace(**0**, **9**, **1000**)  line\_y = -theta\_cap\_cvx[**2**] / theta\_cap\_cvx[**1**] - theta\_cap\_cvx[**0**] / theta\_cap\_cvx[**1**] \* line\_x  plt.scatter(line\_x,line\_y, c ="black", linewidths=**0.1**)  plt.show()  mu\_class0 = np.mean(class0.T, axis=**1**)  sigma\_class0 = np.cov(class0.T,bias=True)  mu\_class1 = np.mean(class1.T, axis=**1**)  sigma\_class1 = np.cov(class1.T,bias=True)  pi0 = class0.size/(class0.size+class1.size)  pi1 = class0.size/(class0.size+class1.size)  sigma\_class0\_inv = np.linalg.inv(sigma\_class0)  sigma\_class1\_inv = np.linalg.inv(sigma\_class1)  log\_det\_sigma\_class0 = np.log(np.linalg.det(sigma\_class0))  log\_det\_sigma\_class1 = np.log(np.linalg.det(sigma\_class1))  **def** **dec\_rule**(x):  x\_class0\_hat = x - mu\_class0  x\_class0\_hat\_t = x\_class0\_hat.transpose()  x\_class1\_hat = x - mu\_class1  x\_class1\_hat\_t = x\_class1\_hat.transpose()  param1\_class0 = -**0.5**\* x\_class0\_hat\_t@ sigma\_class0\_inv @ x\_class0\_hat  param1\_class1 = -**0.5**\*x\_class1\_hat\_t @ sigma\_class1\_inv @ x\_class1\_hat  param2\_class0 = -**0.5**\*log\_det\_sigma\_class0  param2\_class1 = -**0.5**\*log\_det\_sigma\_class1  **return** (param1\_class0 + param2\_class0 - param1\_class1 - param2\_class1)  X1 = np.linspace(-**5**, **10**, **100**)  X2 = np.linspace(-**5**, **10**, **100**)  Z = np.zeros((**100**,**100**))  **for** i **in** range(**0**,len(X1)):  **for** j **in** range(**0**,len(X2)):  x = [X1[i], X2[j]]  **if**(dec\_rule(x) > **0**):  Z[i,j] = **1**  **print**(Z)  [X, Y] = np.meshgrid(X1, X2)  fig, ax = plt.subplots(**1**, **1**)  plt.scatter(class0[:,**0**], class0[:,**1**], edgecolor ="blue", marker ="o")  plt.scatter(class1[:,**0**], class1[:,**1**], c ="red", marker =".")  ax.contour(X, Y, Z)  plt.show() |

Chart, scatter chart

Description automatically generated

Chart, scatter chart

Description automatically generated

**Exercise 3**

**a)**

K[47:52,47:52] =

[[1.00000000e+00 5.64475274e-04 1.29143636e-03 3.10617742e-04

2.14774769e-03]

[5.64475274e-04 1.00000000e+00 4.77708365e-03 1.73395918e-04

6.13460716e-03]

[1.29143636e-03 4.77708365e-03 1.00000000e+00 5.03177733e-06

1.54933077e-04]

[3.10617742e-04 1.73395918e-04 5.03177733e-06 1.00000000e+00

2.70068814e-02]

[2.14774769e-03 6.13460716e-03 1.54933077e-04 2.70068814e-02

1.00000000e+00]]

Letter

Description automatically generated

c)

first two values of alpha\_cap by cvx method :

[[-0.74267438]

[-0.8716697 ]]

d)

Chart, scatter chart

Description automatically generated

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | **import** **numpy** **as** **np**  **from** **numpy.matlib** **import** repmat  **import** **matplotlib.pyplot** **as** **plt**  **import** **cvxpy** **as** **cp**  **import** **csv**  class0 = []  class1 = []  # Reading csv file for male data  **with** open("data/homework4\_class0.txt", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=' ')  **for** row **in** reader:  row = [i **for** i **in** row **if** i != '']  class0.append(list(np.float\_(row)))  class0 = np.array(class0)  csv\_file.close()  **print**(class0.shape)  **with** open("data/homework4\_class1.txt", "r") **as** csv\_file:  reader = csv.reader(csv\_file, delimiter=' ')  **for** row **in** reader:  row = [i **for** i **in** row **if** i != '']  class1.append(list(np.float\_(row)))  class1 = np.array(class1)  csv\_file.close()  **print**(class1.shape)  #least squares  N = class0.shape[**0**] + class1.shape[**0**]  d = **3**  x = np.vstack((class0[:,**0**:**2**],class1[:,**0**:**2**]))  X = np.column\_stack((x, np.ones(N))) #consider the basis function as 1 + x1 + x2  y = np.vstack((np.zeros((class0.shape[**0**],**1**)),np.ones((class1.shape[**0**],**1**))))  lambd = **0.0001**  one\_transpose = np.ones((**1**,N))  K = np.zeros((**100**,**100**))  **for** i **in** range(**0**,N):  **for** j **in** range(**0**,N):  K[i,j] = np.exp(-np.linalg.norm((x[i]-x[j]),**2**))  **print**(K[**47**:**52**,**47**:**52**])  alpha = cp.Variable((N,**1**))  log\_likelihood = cp.sum(cp.multiply(y,K**@alpha**)) - cp.sum(cp.log\_sum\_exp(cp.hstack([np.zeros((N,**1**)),K**@alpha**]),axis=**1**))  prob = cp.Problem(cp.Maximize(log\_likelihood/N - lambd \* cp.quad\_form(alpha, K)))  prob.solve()  alpha\_cap\_cvx = alpha.value  #print(alpha\_cap\_cvx)  **print**(f"first two values of alpha\_cap by new cvx method :**\n** {alpha\_cap\_cvx[:2]}")  X1 = np.linspace(-**5**, **10**, **100**)  X2 = np.linspace(-**5**, **10**, **100**)  Z = np.zeros((**100**,**100**))  **for** i **in** range(**100**):  **for** j **in** range(**100**):  data = repmat( np.array([X1[i], X2[j], **1**]).reshape((**1**,**3**)), N, **1**)  s = data - X  ks = np.exp(-np.sum(np.square(s), axis=**1**))  Z[i,j] = np.dot(alpha\_cap\_cvx.T, ks).item()  plt.figure(figsize=(**10**,**10**))  plt.scatter(class0[:,**0**], class0[:,**1**], edgecolor ="blue", marker ="o")  plt.scatter(class1[:,**0**], class1[:,**1**], c ="red", marker =".")  plt.contour(X1, X2, Z>**0.5**, linewidths=**1**, colors='k')  plt.legend(['Class0', 'Class1'])  plt.title('Kernel Method')  plt.show() |