Math 7243 Machine Learning - Homework 3

For programming questions, you can only use numpy library.

Question 1. Softmax regression Recall the setup of logistic regression: We assume that the posterior probability is of the form

$$p(Y = 1|\vec{x}) = \frac{1}{1 + e^{-\beta^T \vec{x}}}$$

This assumes that $Y|X$ is a Bernoulli random variable. We now turn to the case where $Y|X$ is a multinomial random variable over K outcomes. This is called softmax regression, because the posterior probability is of the form

$$p(Y = k|\vec{x}) = \frac{e^{\beta_k^T \vec{x}}}{\sum_{j=1}^K e^{\beta_j^T \vec{x}}}$$

which is called the softmax function. Assume we have observed data $D = \{\vec{x}^{(i)}, y^{(i)}\}_{i=1}^N$. Our goal is to learn the weight $\beta_1, ..., \beta_K$.

(1) Find the negative log likelihood of the data $l(\beta_1, ..., \beta_K) = -\log L(\beta_1, ..., \beta_K) = -\log P(Y|X)$

$$-\log \mathbb{P}(Y|X) = -\log \prod_{i=1}^N \mathbb{P}(y_i|x_i) = -\log \prod_{i=1}^N \prod_{k=1}^K \left(\frac{e^{\beta_k^T x_i}}{\sum_{j=1}^K e^{\beta_j^T x_i}} \right)^{1\{y_i=k\}}$$

$$= -\sum_{i=1}^N \sum_{k=1}^K 1\{y_i = k\} \left(\beta_k^T x_i - \log \left(\sum_{j=1}^K e^{\beta_j^T x_i} \right) \right)$$

$$= -\sum_{i=1}^N \sum_{k=1}^K 1\{y_i = k\} \beta_k^T x_i + \sum_{i=1}^N \log \left(\sum_{j=1}^K e^{\beta_j^T x_i} \right)$$

(2) We want to minimize the negative log likelihood. To combat overfitting, we put a regularizer on the objective function. Find the **gradient** w.r.t. β_k of the regularized objective

$$l(\beta_1, ..., \beta_K) + \lambda \sum_{k=1}^K ||\beta_k||^2$$

$$\nabla_{\beta_k} - \log \mathbb{P}(Y|X) = 2\lambda\beta_k - \sum_{i=1}^N 1\{y_i = k\} x_i + \sum_{i=1}^N \frac{e^{\beta_k^T x_i}}{\sum_{j=1}^K e^{\beta_j^T x_i}} x_i$$

Note that we can use the definition of $\mu_k(x_i)$ here to save a bunch of writing.

$$= 2\lambda\beta_k + \sum_{i=1}^N (\mu_k(x_i) - 1\{y_i = k\}) x_i$$

(3) State the gradient updates for both batch gradient descent and stochastic gradient descent.

Batch gradient descent:

$$\beta_k^{(t+1)} = \beta_k^{(t)} - \eta \left(2\lambda \beta_k^{(t)} + \sum_{i=1}^{N} \left(\mu_k(x_i) - 1\{y_i = k\} \right) x_i \right)$$

Stochastic gradient descent:

$$\beta_k^{(t+1)} = \beta_k^{(t)} - \eta \left(2\lambda \beta_k^{(t)} + \left(\mu_k(x_i) - 1\{y_i = k\} \right) x_i \right)$$

Question 2. Logistics Regression Consider the categorical learning problem consisting of a data set with two labels:

Label 1:

X_1	3.81	0.23	3.05	0.68	2.67
X_2	-0.55	3.37	3.53	1.84	2.74

Label 2:

X_1	-2.04	-0.72	-2.46	-3.51	-2.05
X_2	-1.25	-3.35	-1.31	0.13	-2.82

(1) Use **gradient descent** to find the logistic regression model

$$p(Y = 1|\vec{x}) = \frac{1}{1 + e^{-\beta^T \vec{x}}}$$

and the boundary. (Plot the boundary, only use numpy and Matplotlib.)

Using the following data matrix to do the logistic regression. (Notice that we need to use labels 0, 1)

X_1	X_2	Y
3.81	-0.55	1
0.23	3.37	1
3.05	3.53	1
0.68	1.84	1
2.67	2.74	1
-2.04	-1.25	0
-0.72	-3.35	0
-2.46	-1.31	0
-3.51	0.13	0
-2.05	-2.82	0

The (matrix notation) of the gradient of the Cross-Entropy cost J can be coded as

```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def grad_cost(theta, x, y):
    z = x.dot(theta)
    gradcost = (1/len(x))*np.matmul(x.T,(sigmoid(z)-y))
    return gradcost
```

Define Gradient Descent function with iterations and learning rate alpha

```
def GradientDescent(x,y, theta, alpha, iteration):
    for i in range(iteration):
        theta_new = theta - alpha*grad_cost(theta,x,y)
        theta = theta_new
    return theta_new
```

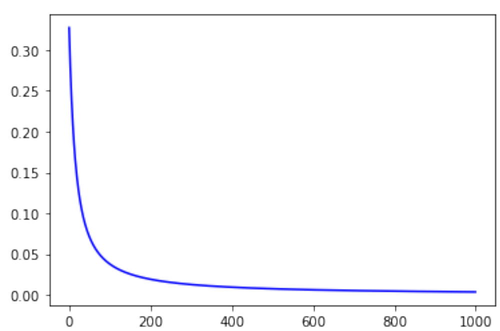
The result of $\vec{\theta}$ depends on your initial value $\vec{\theta})_0$, number iterations, and learning rate α. With $\vec{\theta})_0 = \vec{0}$, $\alpha = 0.02$, and and 1000 iterations, we get our $\vec{\theta}$:

[-0.04617983, -1.37920924, -1.25274956]

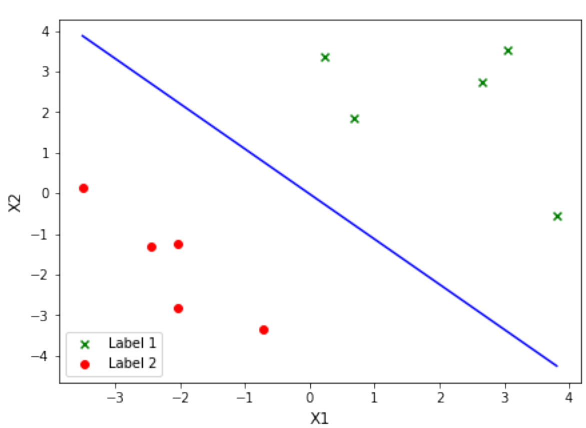
(Your answer may very different from this. But after divide θ_2, the answer should be similarly. Or the boundary graph should be similarly.)

If you want, you can also recording the Cross-entropy cost values and plot them. The cross entropy function can be defined as:

```
def CELoss(x,y,theta):
    z = y*x.dot(theta)
    CE=np.matmul(y.T,np.log(sigmoid(z)))+np.matmul((np.ones(y.shape)-y).T,np.log((np.ones(sigmoid(z).shape))))
    return -(1/len(x))*CE
```



The boundary $\theta_0 + \theta_1 X_1 + \theta_2 X_2 = 0$ can be plotted using $plt.plot(X1, (-X1 * \theta_1 - \theta_0)/\theta_2, color = "blue")$ (Here, you only need to plot two points for X_1, i.e, the min and the max.)



(2) Try **quadratic** Logistics Regression method for this question and obtain an quadratic boundary. (bonus) (Hint: this means to use new features: X_1, X_2, X_1^2, X_1X_2, X_2^2.)

Using the following data matrix to do the LDA again:

X_1	X_2	X_1^2	X_1X_2	X_2^2	Y
3.81	-0.55	3.81^2	(3.81)(-0.55)	$(-0.55)^2$	1
0.23	3.37				1
3.05	3.53	$\vdots$	$\vdots$	$\vdots$	1
0.68	1.84				1
2.67	2.74				1
-2.04	-1.25				0
-0.72	-3.35				0
-2.46	-1.31				0
-3.51	0.13				0
-2.05	-2.82				0

Redo the calculation based on the new data matrix. We have the $\vec{\theta}^T = [\theta_0\ \theta_1\ \theta_2\ \theta_3\ \theta_4\ \theta_5\ ]$:
array([-0.01614066, -1.33955452, -1.23265001, 0.02176921, 0.20651087, -0.11120619])
(Again, your answer may very different from this. But after divide θ_2, the answer should be similarly. Or the boundary graph should be similarly.)
The boundary $\theta_0 + \theta_1X_1 + \theta_2X_2 + \theta_3X_1^2 + \theta_4X_1X_2 + \theta_5X_2^2 = 0$

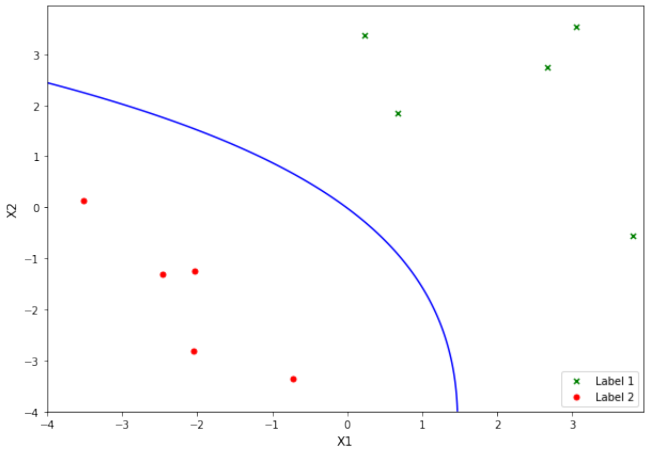
Remark: You may get the polynomial feature by basic coding: numpy.c_[x, x1*x1, x1*x2,x2*x2] to add columns. If allow to use scikit-learn in labs, we can use sklearn.preprocessing (See CVBootstrap.ipynb in lecture notes)

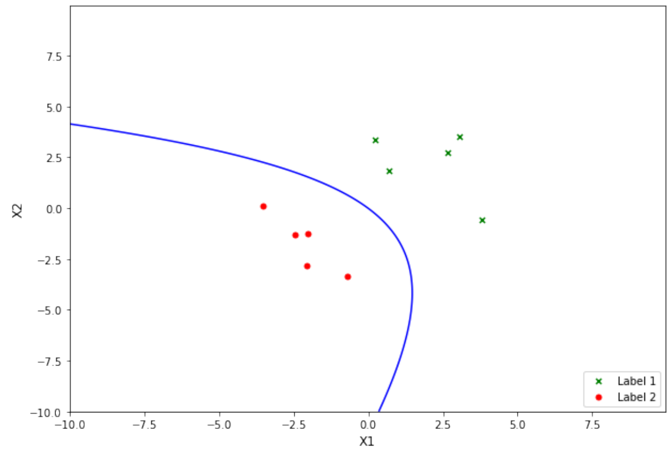
```
from sklearn.preprocessing import PolynomialFeatures
# Quadratic
poly = PolynomialFeatures(degree=2)
x_poly = poly.fit_transform(x)

```

Graphing: You may use the following code to draw the graph:

```
X, Y = np.meshgrid(np.arange(-4, 4, 0.05),np.arange(-4, 4, 0.05))
plt.contour(X, Y,
-0.01614066-1.33955452*X-1.23265001*Y+0.02176921*X*X+0.20651087*X*Y-0.11120619*Y*Y ,
    [0])
plt.show()
```

The same drawing in different ranges.





Question 3. - Linear Discriminant Analysis: Consider the categorical learning problem consisting of a data set with two labels:

Label 1:

X_1	3.81	0.23	3.05	0.68	2.67
X_2	-0.55	3.37	3.53	1.84	2.74

Label 2:

X_1	-2.04	-0.72	-2.46	-3.51	-2.05
X_2	-1.25	-3.35	-1.31	0.13	-2.82

a) For each label above, the data follow a multivariate normal distribution Normal(μ_i, Σ) where the covariance Σ is the same for both label 1 and for label 2. Fit a pair of Guassian discriminant functions to the labels by computing the covariances, means, and proportions of datapoints as discussed in the Linear Discriminant Analysis section. You may use a computer, but you should **not** use an LDA solver. You should report the values for μ_i and Σ.

$$\mu_1 = \begin{pmatrix} 2.088 \\ 2.186 \end{pmatrix}, \mu_2 = \begin{pmatrix} -2.156 \\ -1.72 \end{pmatrix}.$$

$$\Sigma = \frac{1}{10-2} \sum_{i=1}^{10} (X^{(i)} - \mu_k)(X^{(i)} - \mu_k)^T = \begin{pmatrix} 1.709575 & -1.23013 \\ -1.23013 & 2.349865 \end{pmatrix}.$$

$$\phi_1 = \phi_2 = \frac{5}{10} = \frac{1}{2}.$$

$$P(X|\text{Label} = 1) = \frac{1}{(2n)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1))$$

$$P(X|\text{Label} = 2) = \frac{1}{(2n)^{n/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x - \mu_2)^T \Sigma^{-1} (x - \mu_2))$$

Suppose you already have the standard data matrices M1 and M2 as our standard form. (each one is a 5 by 2 matrix) The code for mean and covariance can be:

```
# means
mu1=M1.mean(0)
mu2=M1.mean(0)
```

```
# covariance matrix
(1/(len(M1)+len(M2)-2))*(np.matmul((M1-M1.mean(0)).T, M1-M1.mean(0))\
+np.matmul((M2-M2.mean(0)).T, M2-M2.mean(0)))

 array([[ 1.709575,  -1.23013 ],
        [-1.23013 ,   2.349865]])
```

b) Give the **formula for the line** forming the discretion boundary.

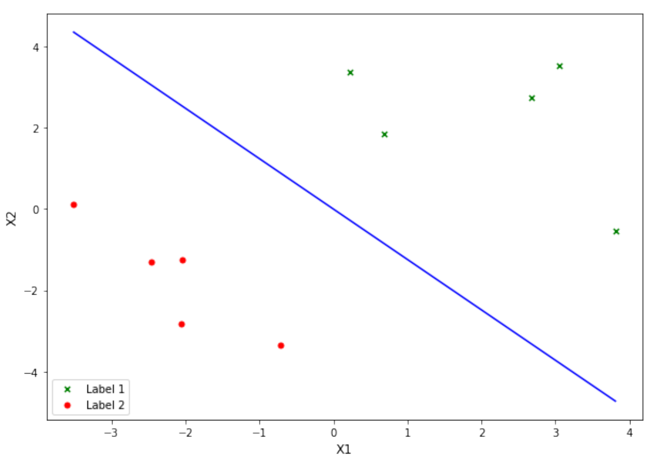
The line forming the discretion boundary is $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ such that $\log \frac{P(\text{Label}=2|X)}{P(\text{Label}=1|X)} = 0$.

$P(\text{Label} = k|X) = \frac{P(X|\text{Label}=k)P(\text{Label}=k)}{P(X)}$.

$$\log P(\text{Label} = k|X) = \log P(X|\text{Label} = k) + \log P(\text{Label} = k) - \log P(X)$$
$$= -\tfrac{1}{2}\log|\Sigma| - \tfrac{1}{2}(X - \mu_k)^T\Sigma^{-1}(X - \mu_k) + \log\phi_k + \text{constant}$$
$$= X^T\Sigma^{-1}\mu_k - \tfrac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\phi_k - \tfrac{1}{2}\log|\Sigma| + \text{constant}$$

Hence, $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ such that $\log P(\text{Label} = 1|X) = \log P(\text{Label} = 2|X)$.

$$X^T\Sigma^{-1}\mu_1 - \tfrac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 = X^T\Sigma^{-1}\mu_2 - \tfrac{1}{2}\mu_2^T\Sigma^{-1}\mu_2$$
$$\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 3.03331817 \\ 2.51817686 \end{pmatrix} - 5.91915147956369 = \begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} -2.86820579 \\ -2.23343298 \end{pmatrix} - 5.012678200684864$$
$$\begin{pmatrix} x_1 & x_2 \end{pmatrix}\begin{pmatrix} 3.03331817 + 2.86820579 \\ 2.51817686 + 2.23343298 \end{pmatrix} = -5.012678200684864 + 5.91915147956369$$
$$5.90152396x_1 + 4.75160984x_2 = 0.9064732788788268$$
$$x_2 = \frac{0.9064732788788268 - 5.90152396x_1}{4.75160984}$$



c) Use the **QDA** method for this question and obtain an quadratic boundary. (Hint, you need to calculate Σ_1 and Σ_2 separately.)

We assume the covariance Σ_1 and Σ_2 for each label are different. In this case,

$$\Sigma_1 = \frac{1}{5-1}\sum_{i=1}^{5}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T = \begin{bmatrix} 2.41602 & -1.202185 \\ -1.202185 & 2.78013 \end{bmatrix}$$

$$\Sigma_2 = \frac{1}{5-1}\sum_{i=1}^{5}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T = \begin{bmatrix} 1.00313 & -1.258075 \\ -1.258075 & 1.9196 \end{bmatrix}$$

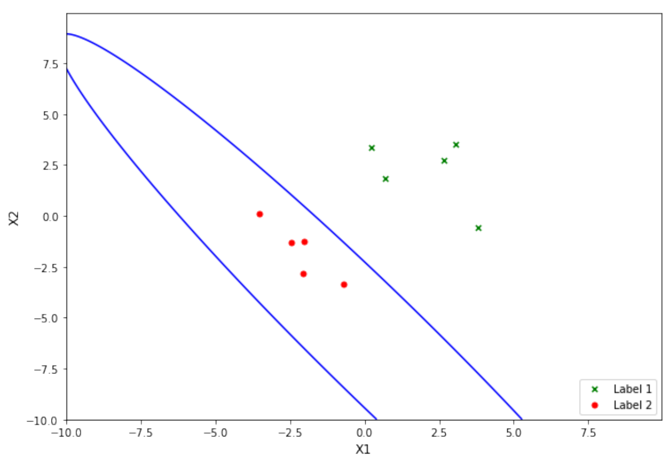
Simplify the calculation with constants, we have the equality

$$-\frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(\vec{x}-\mu_1)^T\Sigma^{-1}(\vec{x}-\mu_1) + \log(\phi_1) = -\frac{1}{2}\log|\Sigma_2| - \frac{1}{2}(\vec{x}-\mu_2)^T\Sigma^{-1}(\vec{x}-\mu_2) + \log(\phi_2)$$

Plug in the information from (1) and Σ_1, Σ_2 we have the quadratic curve

$$2.536x^2 + 3.441xy + 19.982x + 1.234y^2 + 14.421y + 26.296 = 0$$

```python
# covariance matries
def covar(x):
    return (1/(len(x)-1))*np.matmul((x-x.mean(0)).T, x-x.mean(0))

sigma_1 = covar(M1)
sigma_2 = covar(M2)
```



To simply the formula, I used the **sympy** library. So I don't have to simply by hand.

```python
def quForm(x,S):
    return np.matmul(np.matmul(x.T, S), x)

from sympy import *

xx, yy  = symbols("xx yy")

simplify(math.log(phi_1)-math.log(phi_2)\
-0.5*quForm((np.array([xx,yy])-mu1), np.linalg.inv(sigma_1))\
+0.5*quForm((np.array([xx,yy])-mu2), np.linalg.inv(sigma_2))\
-0.5*math.log(np.linalg.det(sigma_1)) + 0.5*math.log(np.linalg.det(sigma_2))).evalf(4)
```

(d) Try quadratic LDA method for this question and obtain an quadratic boundary. (bonus)

Using the following data matrix to do the LDA again:

X_1	X_2	X_1^2	X_1X_2	X_2^2	Y
3.81	-0.55	3.81^2	(3.81)(-0.55)	$(-0.55)^2$	1
0.23	3.37				1
3.05	3.53	$\vdots$	$\vdots$	$\vdots$	1
0.68	1.84				1
2.67	2.74				1
-2.04	-1.25				0
-0.72	-3.35				0
-2.46	-1.31				0
-3.51	0.13				0
-2.05	-2.82				0

and obtain the formula

$$\vec{x}^T\Sigma^{-1}\vec{\mu}_1 - \frac{1}{2}\vec{\mu}_1^T\Sigma^{-1}\vec{x} + \log\phi_1 = \vec{x}^T\Sigma^{-1}\vec{\mu}_2 - \frac{1}{2}\vec{\mu}_2^T\Sigma^{-1}\vec{x} + \log\phi_2$$

Notice that our calculation is in dimension 5.

$$-2.34x^2 - 3.187xy + 20x - 0.914y^1 + 15y + 26.223 = 0$$

Again, we can use plt.contour() to draw the graph.

```
X,Y = np.meshgrid(np.arange(-50, 50, 0.05),np.arange(-50, 50, 0.05))
plt.contour(X,Y, -2.34*X*X-3.187*X*Y+20*X-0.914*Y*Y+15*Y+26.223, [0], colors="blue")

plt.legend(loc='lower right')
ax1.set_xlabel('X1')
ax1.set_ylabel('X2')
fig.set_size_inches(10, 7)
plt.show()
```