Question 1

Logistic Regression for multiclass classification

In [41]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [42]:
```
dataset = pd.read_csv('Iris.csv', header=0)
dataset.sample(n=5)
```

Out[42]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
149	150	5.9	3.0	5.1	1.8	Iris-virginica
129	130	7.2	3.0	5.8	1.6	Iris-virginica
102	103	7.1	3.0	5.9	2.1	Iris-virginica
5	6	5.4	3.9	1.7	0.4	Iris-setosa
70	71	5.9	3.2	4.8	1.8	Iris-versicolor

In [43]:
```
x = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
x = np.hstack((np.ones((x.shape[0], 1)), x)) #[1 x]
print(x.shape)
```

```
(150, 6)
```

In [44]:
```
#since multiclass regression has different outcomes and we need to identify unique s
y_class = y.unique()
# y_class = np.insert(y_class, 0, 10)
y_class
```

Out[44]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [45]:
```
#in the given dataset, replace the Species column with the numerical value that refe
Y = np.zeros((y.shape[0], len(y_class)))
print(Y.shape)
for i in range(len(Y)):
    for j in range(len(y_class)):
        if y_class[j] == y[i]:
            Y[i][j] = 1
```

```
(150, 3)
```

In [46]:
```
train_x, test_x, train_y, test_y = train_test_split(x, Y, train_size=0.8, shuffle=Tr
```

In [47]:
```
print(train_x.shape)
print(test_x.shape)
print(train_y.shape)
print(test_y.shape)
```

```
(120, 6)
(30, 6)
(120, 3)
(30, 3)
```

In [48]:
```
theta = np.zeros((x.shape[1], len(y_class)))
theta.shape
```

Out[48]:
```
(6, 3)
```

In [49]:
```
def sigmoid(x, theta):
    return 1/(1+np.exp(-np.dot(x, theta)))
```

In [50]:
```
def multiclass_logistic_regression(x, y, theta, alpha, iterations):
    m = x.shape[0]
    for _ in range(iterations):
        prec_y = sigmoid(x, theta)
        theta = theta - (alpha/m)*np.dot(x.T, prec_y -y)
    return theta
```

In [51]:
```
theta = multiclass_logistic_regression(train_x, train_y, theta, 0.0002, 70000)
```

In [52]:
```
prediction = sigmoid(test_x, theta)
prediction
```

Out[52]:
```
array([[2.13743706e-02, 5.25830171e-01, 1.04965366e-01],
       [1.65184315e-06, 1.23879142e-01, 9.86888110e-01],
       [9.90104492e-01, 1.40687376e-01, 5.54648280e-04],
       [2.93594031e-02, 5.17487766e-01, 1.04018403e-01],
       [2.48065598e-03, 4.61045228e-01, 3.35372134e-01],
       [9.98637057e-01, 2.13703306e-01, 1.45623929e-04],
       [5.14627031e-03, 3.55095079e-01, 3.33900260e-01],
       [3.08704276e-01, 6.08068220e-01, 1.01606286e-02],
       [6.42140879e-02, 3.52351103e-01, 6.73759513e-02],
       [1.10065600e-05, 3.32561448e-01, 9.49605577e-01],
       [2.13902430e-02, 4.43969893e-01, 1.41755634e-01],
       [3.51479224e-05, 7.32783619e-01, 7.75527218e-01],
       [2.94979506e-01, 7.02571206e-01, 7.66048039e-03],
       [1.21392011e-02, 6.54821635e-01, 1.31649831e-01],
       [9.78852492e-01, 1.33595455e-01, 9.12113881e-04],
       [7.07754728e-01, 7.41663527e-02, 8.64058075e-03],
       [2.29134258e-04, 4.38996846e-01, 7.70869753e-01],
       [2.21692969e-04, 5.29343941e-01, 6.69256073e-01],
       [8.41725126e-01, 9.35453997e-02, 4.08448385e-03],
       [1.66157551e-06, 1.73032741e-01, 9.90880232e-01],
       [6.58027485e-06, 3.07006332e-01, 9.64857734e-01],
       [2.14054880e-05, 2.42479213e-01, 9.53633179e-01],
       [9.70058104e-01, 2.03831001e-01, 1.45323324e-03],
       [1.22309198e-03, 3.71151038e-01, 5.55829296e-01],
       [1.37857428e-06, 1.66958830e-01, 9.89756169e-01],
       [4.84315336e-02, 5.79865412e-01, 6.89845560e-02],
       [2.60873240e-02, 4.32302225e-01, 6.44489166e-02],
       [1.09525859e-06, 1.16904746e-01, 9.92194548e-01],
       [2.24951365e-05, 2.00569993e-01, 9.31499560e-01],
       [6.63737125e-01, 8.29168155e-02, 1.29332432e-02]])
```

In [60]:
```
test_y
```

Out[60]:
```
array([[0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

In [62]:
```
error = (np.round(prediction) == test_y)
error
```

Out[62]:
```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True, False,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True, False, False],
       [ True,  True,  True],
       [ True,  True,  True],
       [ True, False,  True],
       [ True,  True,  True],
```

```
       [ True,  True,  True],
       [ True,  True,  True]])
```

In [58]:
```
s = len(y_class)
error_prec = 0
for i in error:
    if np.sum(i) != s:
        error_prec += 1
print(error_prec)
```

```
8
```

In [59]:
```
percentage_error = (error_prec/len(prediction))*100
percentage_error
```

Out[59]:
```
26.666666666666668
```

In [ ]:
```

```