# Q2

March 12, 2019

## 1 Logistic Regression on Marks Data

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.utils import shuffle
```

```
In [2]: df = pd.read_csv('marks.csv', index_col=0)
        df = shuffle(df)
        df.head()
```

```
Out[2]:        marks1     marks2  selected
        5   79.032736  75.344376         1
        63  56.253818  39.261473         0
        45  51.047752  45.822701         0
        21  67.372028  42.838438         0
        78  50.458160  75.809860         1
```

```
In [3]: df.shape
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 5 to 17
Data columns (total 3 columns):
marks1      100 non-null float64
marks2      100 non-null float64
selected    100 non-null int64
dtypes: float64(2), int64(1)
memory usage: 3.1 KB
```

```
In [4]: Y = df['selected']
        X = df.drop(['selected'],axis=1)
```

```
In [5]: print(X.shape)
        print(Y.shape)
```

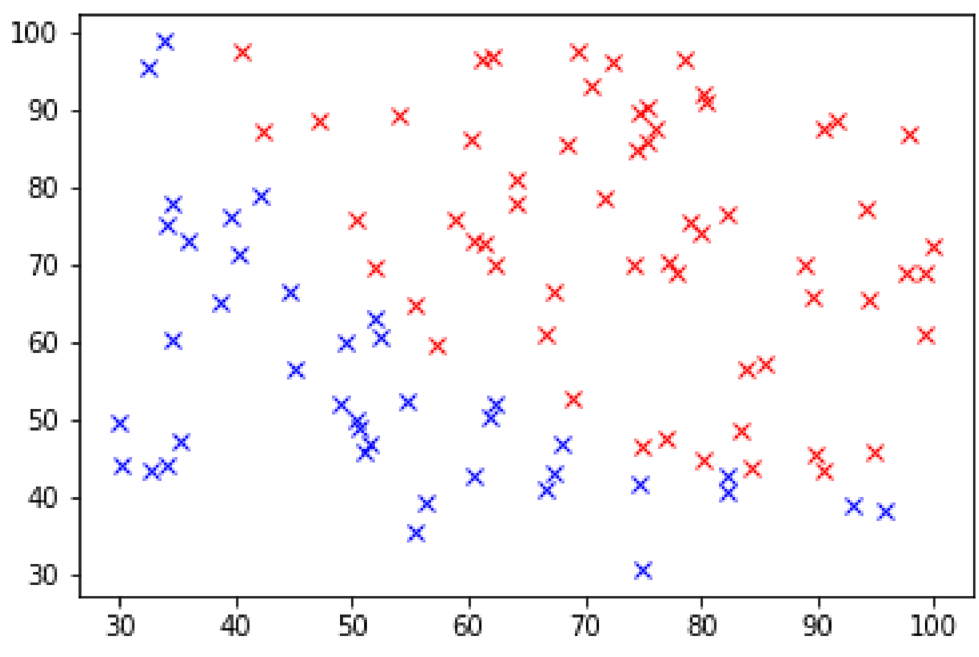
```
(100, 2)
(100,)
```

### 1.1 Data Preprocessing

```
In [6]: Y = (np.array(Y)).reshape(Y.shape[0],1)
        print(Y.shape)
        X = np.array(X)
        X = np.c_[np.ones(X.shape[0]),np.array(X)]
        print(X.shape)
        print(X[:5])
```

```
(100, 1)
(100, 3)
[[ 1.          79.03273605 75.34437644]
 [ 1.          56.2538175  39.26147251]
 [ 1.          51.04775177 45.82270146]
 [ 1.          67.37202755 42.83843832]
 [ 1.          50.4581598  75.80985953]]
```

```
In [7]: for i in range(X.shape[0]):
            if Y[i]==1:
                plt.plot(X[i,1],X[i,2],'rx')
            else:
                plt.plot(X[i,1],X[i,2],'bx')
        plt.show()
```




```
In [8]: #Normalising Inputs(2D input)
        def normalise(inp):
```

```
        return np.array((inp-inp.mean())/inp.std())
    X = normalise(X)
```

```
In [9]: # Splitting data in train and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3)
        print(X_train.shape, y_train.shape)
        print(X_test.shape, y_test.shape)
```

```
(70, 3) (70, 1)
(30, 3) (30, 1)
```

### 1.2 Defining functions

```
In [10]: # Sigmoid Function
         def sigmoid(z):
             return 1.0 / (1 + np.exp(-z))
```

```
In [11]: # Cost Function with Regularization
         def cost(X,Y,theta,_lambda=0.1):
             m = len(Y)
             h = sigmoid(X.dot(theta))
             reg = (_lambda/(2 * m)) * np.sum(theta**2)
             return (1 / m) * (-Y.T.dot(np.log(h)) - (1 - Y).T.dot(np.log(1 - h))) + reg
```

```
In [12]: # Regularized gradient function
         def derivative(X,Y,theta,_lambda = 0.1):
             m, n = X.shape
             h = sigmoid(X.dot(theta))
             reg = _lambda * theta /m
             return ((1 / m) * X.T.dot(h - Y)) + reg
```

```
In [13]: # Training function
         def train(X, Y, alpha, iterations = 500000, _lambda=0.1):
             costs = np.empty([iterations])
             i = 0
             m = X.shape[0]
             theta = np.random.randn(X.shape[1],1)

             while i < iterations:
                 costs[i] = cost(X, Y, theta, _lambda)
                 theta = theta - (alpha / m) * (derivative(X, Y, theta, _lambda))
                 i = i + 1
             return theta, costs
```

```
In [14]: def test(X,Y,theta):
             pred = sigmoid(X.dot(theta))
             counts = 0
             a, b = X.shape
```

```
for i in range(a):
    if pred[i] > 0.5:
        if int(Y[i]) == 1:
            counts = counts + 1
    else:
        if int(Y[i]) == 0 :
            counts = counts + 1

print('Accuracy:',counts/a * 100)
```

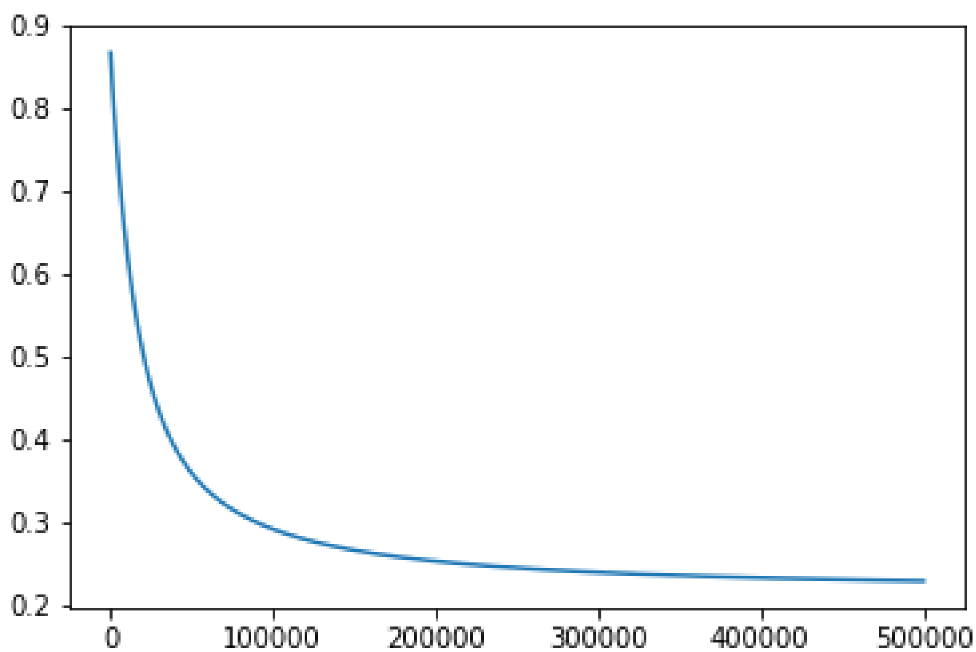
### 1.3 Training without Regularization (lambda=0)

```
In [15]: theta, costs = train(X_train, y_train, 0.05, _lambda=0)
         print(theta)
```

```
[[3.89083734]
 [5.1070446 ]
 [4.88688844]]
```

```
In [16]: plt.plot(range(len(costs)),costs)
```

```
Out[16]: [<matplotlib.lines.Line2D at 0x7f1e6ab8d1d0>]
```




```
In [17]: test(X_test,y_test,theta)
```

```
Accuracy: 96.66666666666667
```

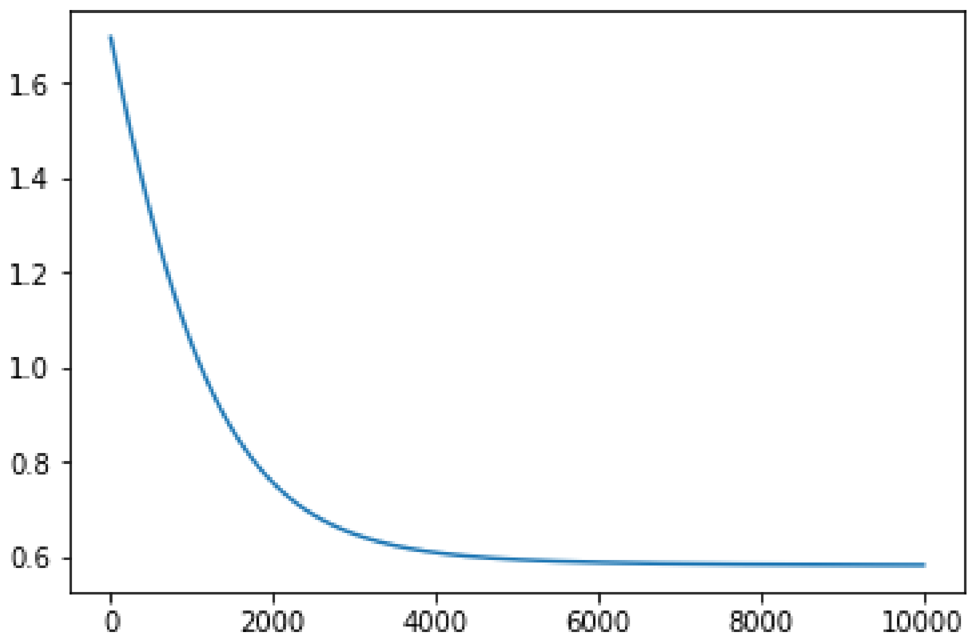
### 1.4 Training with Regularization (lambda = 10)

```
In [18]: theta, costs = train(X_train, y_train, 0.05, iterations = 10000, _lambda=10)
         print(theta)
```

```
[[0.17011868]
 [0.6449835 ]
 [0.51575106]]
```

```
In [19]: plt.plot(range(len(costs)),costs)
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x7f1e6ab6d400>]
```




```
In [20]: test(X_test,y_test,theta)
```

```
Accuracy: 73.33333333333333
```

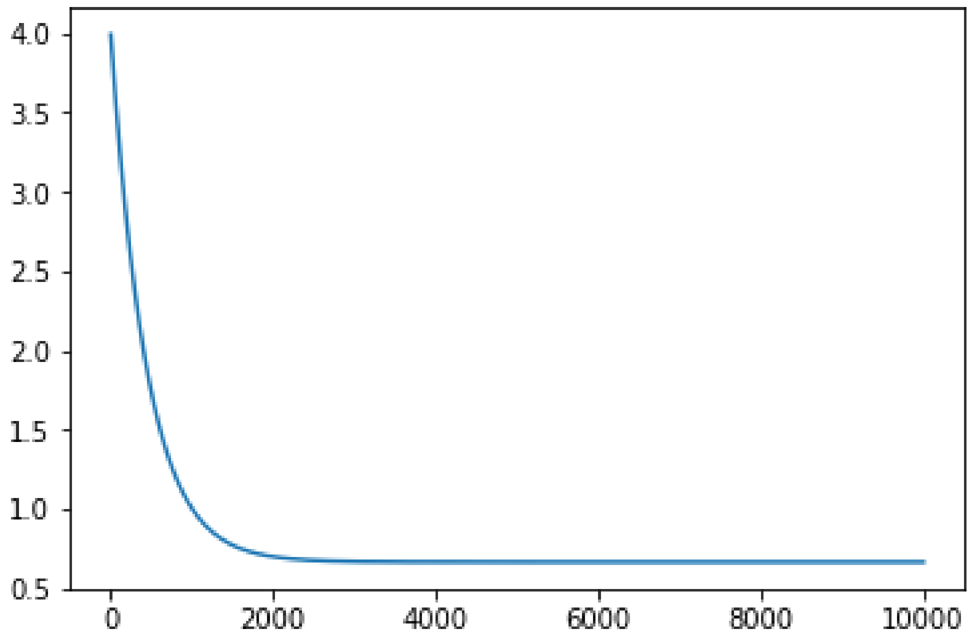
### 1.5 lambda = 100

```
In [21]: theta, costs = train(X_train, y_train, 0.05, iterations = 10000, _lambda=100)
         print(theta)
```

```
[[-0.04430317]
 [ 0.11709424]
 [ 0.11591069]]
```

In [22]:
```
plt.plot(range(len(costs)),costs)
```

Out[22]: [<matplotlib.lines.Line2D at 0x7f1e6aac4da0>]




In [23]:
```
test(X_test,y_test,theta)
```

```
Accuracy: 60.0
```

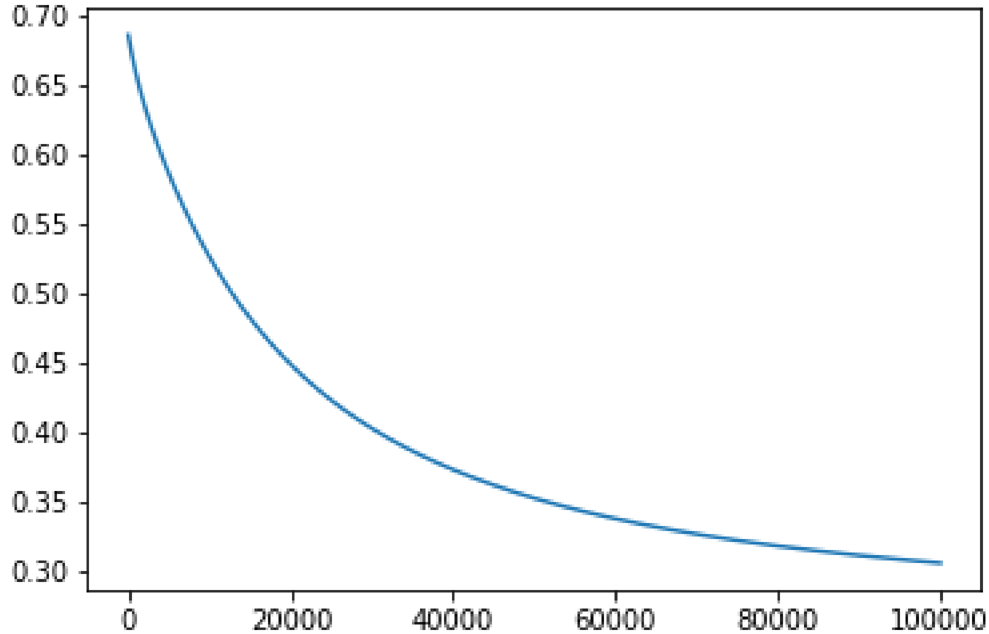
### 1.6 lambda = 0.1

In [24]:
```
theta, costs = train(X_train, y_train, 0.05, iterations = 100000, _lambda=0.1)
print(theta)
```

```
[[1.96763873]
 [2.6855052 ]
 [2.6459638 ]]
```

In [25]:
```
plt.plot(range(len(costs)),costs)
```

Out[25]: [<matplotlib.lines.Line2D at 0x7f1e6aaad0f0>]




```
In [26]: test(X_test,y_test,theta)
```

```
Accuracy: 93.33333333333333
```