06-claret

October 28, 2018

1 Partical Work 06 - Logistic Regression

- Author: *Romain Claret*
- Due-date: *29.10.2018*

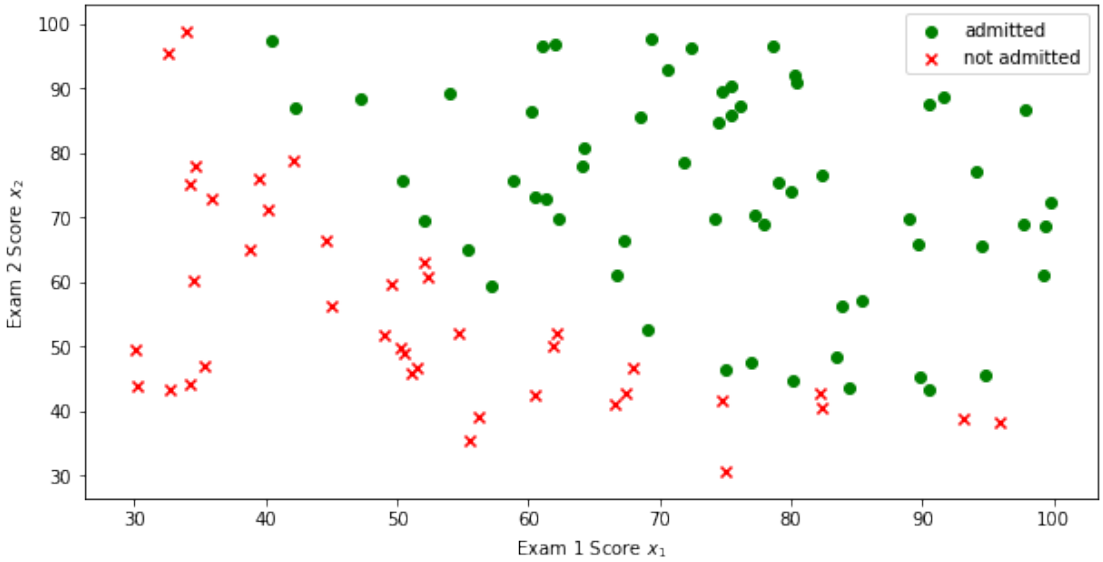
1.1 Exerice 1 - Classification to predict student admission

1.1.1 a) Logistic regression classifier with linear decision boundary

a)

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline


        col_id = ['x1', 'x2', 'y']
        data_train = pd.read_csv('student-dataset-train.csv', names=col_id)
        #print(data_train.head(3))

        x1 = data_train['x1'].values
        x2 = data_train['x2'].values
        y = data_train['y'].values
        N = len(y)

        X = np.matrix([np.ones(len(y)), x1, x2]).T

        x1_pos = [x1[i] for i in range(N) if y[i] == 1]
        x2_pos = [x2[i] for i in range(N) if y[i] == 1]
        x1_neg = [x1[i] for i in range(N) if y[i] == 0]
        x2_neg = [x2[i] for i in range(N) if y[i] == 0]

        plt.figure(figsize=(10, 5))
        plt_pos = plt.scatter(x1_pos, x2_pos, marker="o", label="pass", color="green")
        plt_neg = plt.scatter(x1_neg, x2_neg, marker="x", label="fail", color="red")
        plt.legend((plt_pos,plt_neg),("admitted","not admitted"),loc='upper right')
        plt.xlabel("Exam 1 Score $x_1$")
```

```
        plt.ylabel("Exam 2 Score $x_2$")
        plt.show()

        print("shape of X:", X.shape)
        print("shape of y:", y.shape)
```




```
shape of X: (100, 3)
shape of y: (100,)
```

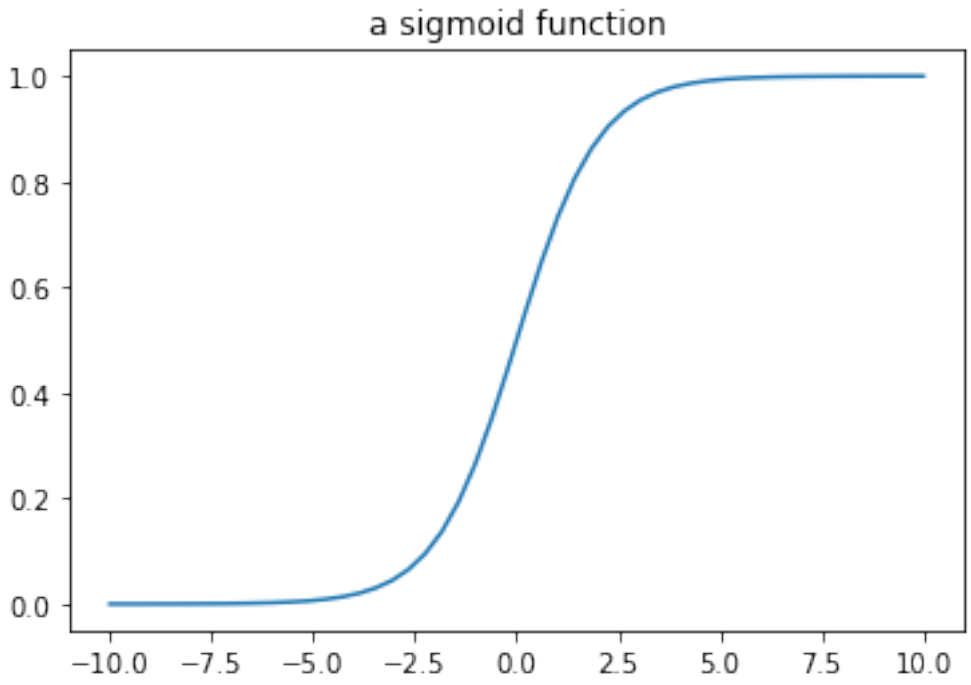
b) Implement a z-norm normalization of the training set. You need to store the normaliza- tion values (,) for later as they will be needed to normalize the test set.

```
In [2]: X_mu = np.mean(X)
        X_sigma = np.std(X)
        X_normalized = (X-X_mu)/X_sigma
        X = X_normalized

        print("mu:", X_mu)
        print("sigma:", X_sigma)
        #print("z-normalisation:", X_normalized)
```

```
mu: 44.28875738181337
sigma: 34.29161449115459
```

c)

```
In [3]: def sigmoid(z):
            return 1.0 / (1.0 + np.exp(-z))

        z = np.linspace(-10, 10)

        plt.plot(z, sigmoid(z))
        plt.title("a sigmoid function")
        plt.show()
```



d)

```
In [4]: def h_theta(X, theta):
            return sigmoid(X.dot(np.matrix(theta).T))
```

e)

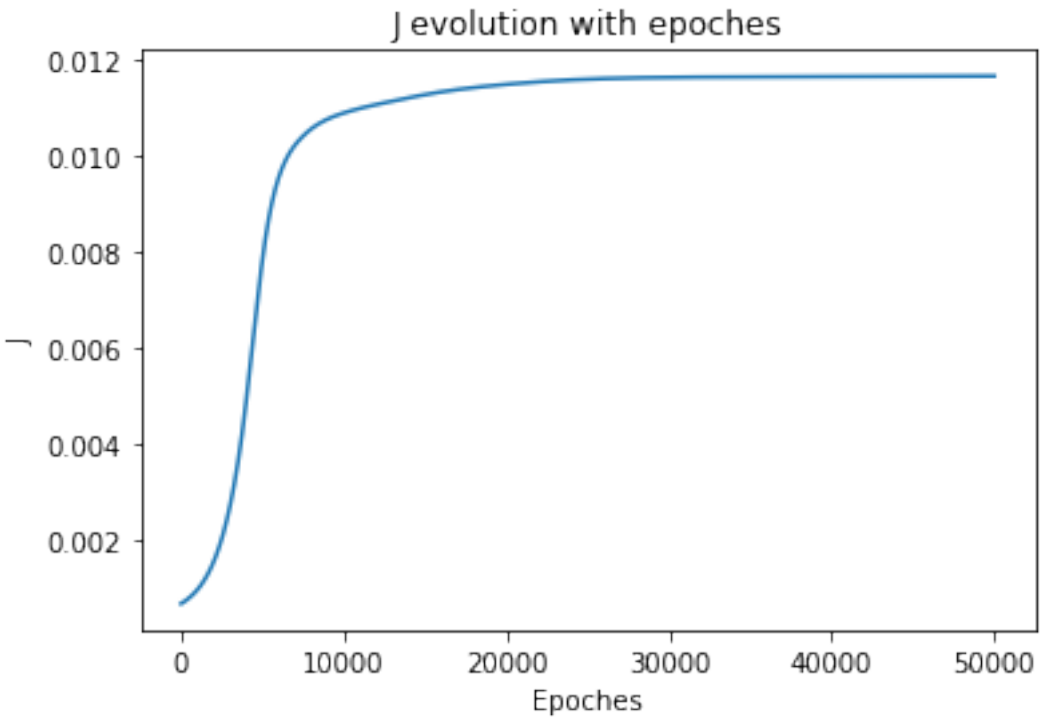
```
In [5]: def J_theta(X, y, theta):
            epsilon = 1e-6
            h = h_theta(X, theta)
            N = X.shape[0]
            tmp = y * np.log(h + epsilon)+(1 - y)*np.log(1 - h + epsilon)
            return np.sum(tmp) / N
```

f)

```
In [6]: def gradientAscend(X, y, learning_rate, num_epoch):
            N, D = X.shape
            theta = np.zeros(D)
            J = []

            for i in range(num_epoch):
                for t in range(D):
                    cost = 0
                    for i in range(N):
                        cost += -J_theta(X, y, theta)
                    cost *= learning_rate * 1.0 / N
                    theta[t] -= cost
                J.append(cost)
            return theta, J
```

g)

```
In [7]: learning_rate = 1e-3
        num_epoch = 50000
        theta, J = gradientAscend(X, y, learning_rate, num_epoch)

        plt.plot(range(num_epoch), J)
        plt.title("J evolution with epoches")
        plt.xlabel("Epoches")
        plt.ylabel("J")
        plt.show()
```



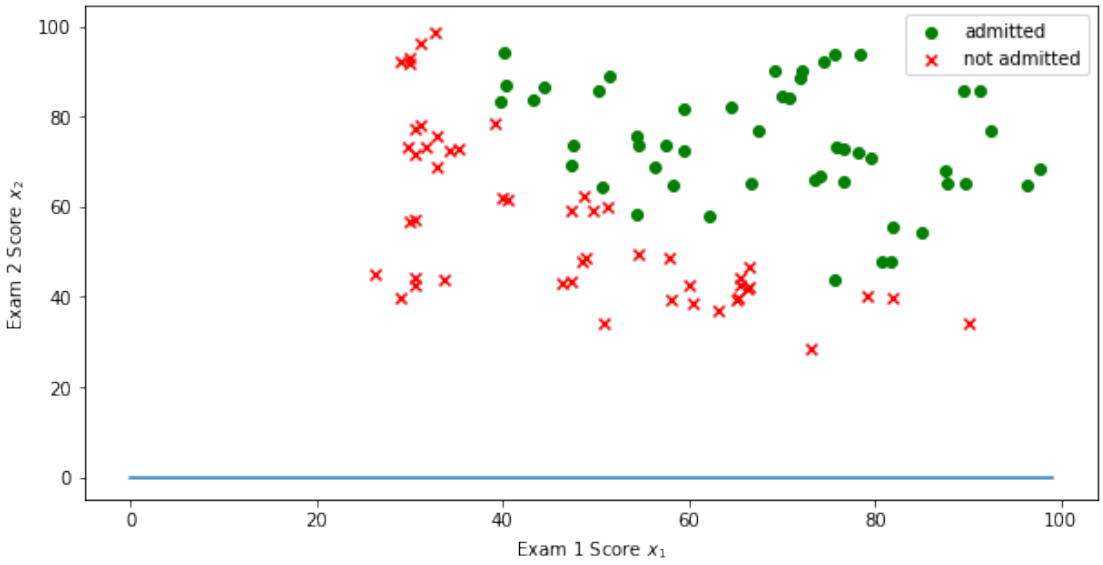
h) This is more than likely wrong

```
In [8]: col_id = ['x1', 'x2', 'y']
        data_test = pd.read_csv('student-dataset-test.csv', names=col_id)

        x1_test = data_test['x1'].values
        x2_test = data_test['x2'].values
        y_test = data_test['y'].values
        N_test = len(y)

        X_test = np.matrix([np.ones(len(y_test)), x1_test, x2_test]).T

        true_prediction = 0
        for i in range(N_test):
            p_1 = h_theta(X_test[i], theta)
            guess = 0
            if p_1 >= 0.5:
                guess = 1
            if y_test[i] == guess:
                true_prediction += 1

        print("Performance: ", true_prediction / N)
        print("Error rate: ", (N_test - true_prediction) / N)
```

```
Performance:  0.5
Error rate:  0.5
```

i) this is so wrong

```
In [9]: x1_pos_test = [x1_test[i] for i in range(N_test) if y_test[i] == 1]
        x2_pos_test = [x2_test[i] for i in range(N_test) if y_test[i] == 1]
        x1_neg_test = [x1_test[i] for i in range(N_test) if y_test[i] == 0]
        x2_neg_test = [x2_test[i] for i in range(N_test) if y_test[i] == 0]

        plt.figure(figsize=(10, 5))

        plt.plot(range(N_test), h_theta(X_test, theta))

        plt_pos_test = plt.scatter(x1_pos_test, x2_pos_test, marker="o", label="pass", color="g
        plt_neg_test = plt.scatter(x1_neg_test, x2_neg_test, marker="x", label="fail", color="r
        plt.legend((plt_pos_test,plt_neg_test),("admitted","not admitted"),loc='upper right')
        plt.xlabel("Exam 1 Score $x_1$")
        plt.ylabel("Exam 2 Score $x_2$")
        plt.show()
```



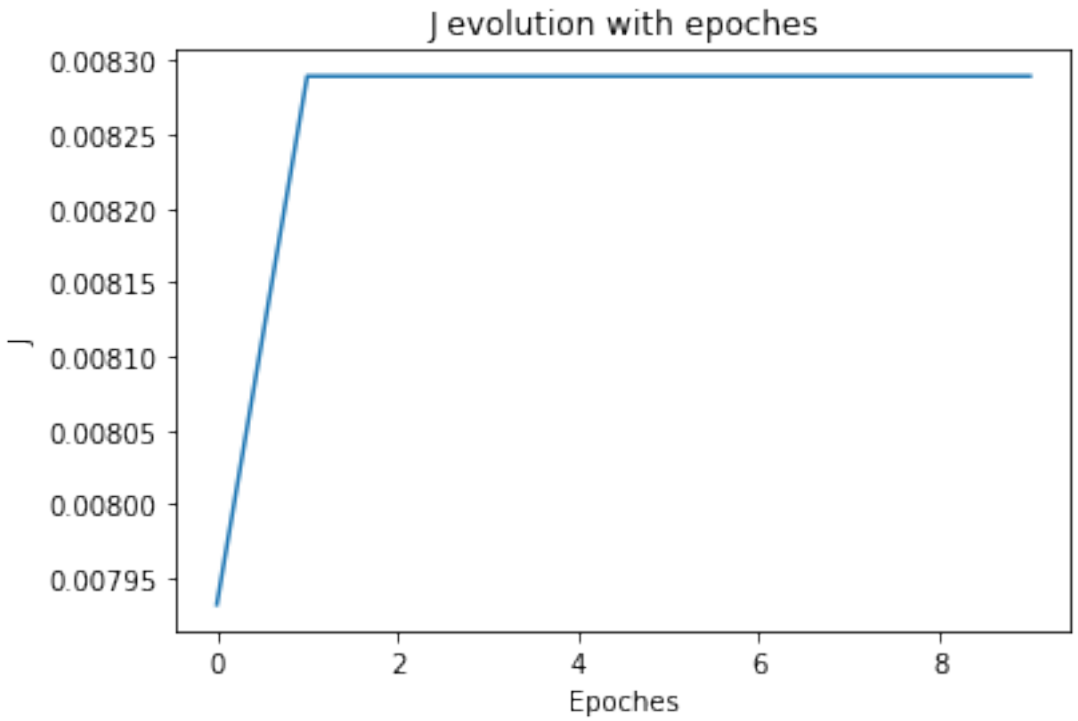
j) It's clear I did some mistakes here.

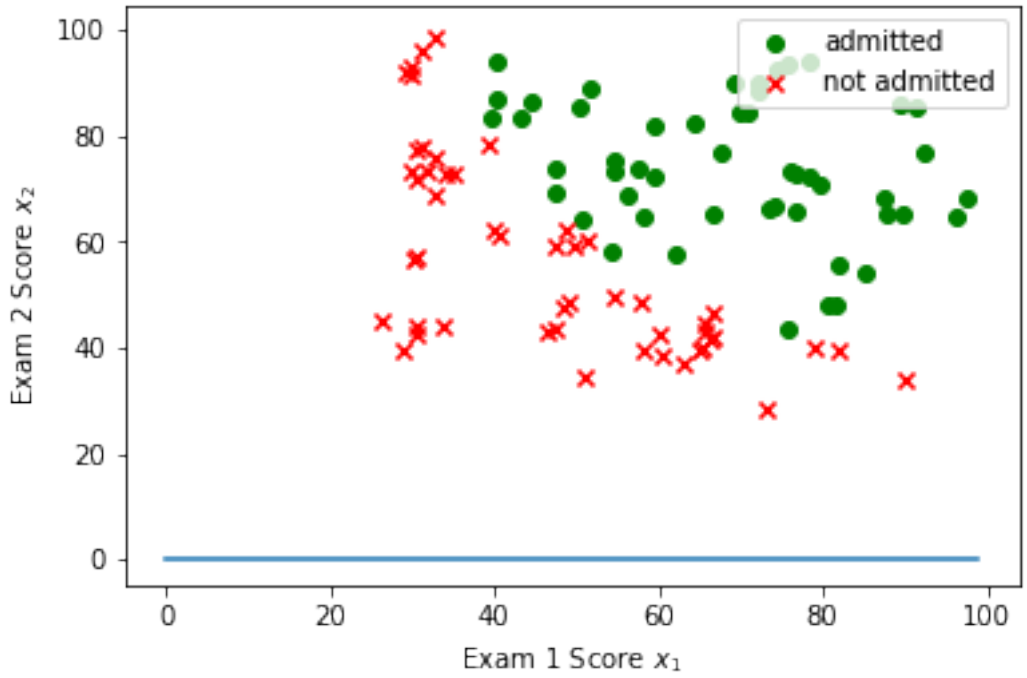
1.1.2 c. Logistic regression classifier with non-linear decision boundary

don't work :(

```
In [10]: x1_complex = data_train['x1'].values
         x2_complex = data_train['x2'].values
         x1_2_complex = np.square(x1)
         x2_2_complex = np.square(x2)
         x1_x2_complex = x1*x2
         y_complex = data_train['y'].values
         N_complex = len(y)

         X_complex = np.matrix([np.ones(len(y_complex)),
                                x1_complex,
                                x2_complex,
                                x1_2_complex,
                                x2_2_complex,
                                x1_x2_complex]).T

         learning_rate_complex = 1e-3
         num_epoch_complex = 10
         theta_complex, J_complex = gradientAscend(X_complex, y_complex, learning_rate_complex

         plt.plot(range(len(J_complex)), J_complex)
         plt.title("J evolution with epoches")
         plt.xlabel("Epoches")
```

```
plt.ylabel("J")
plt.show()

#plt.figure(figsize=(10, 5))

plt.plot(range(N_test), h_theta(X_complex, theta_complex))

plt_pos_test = plt.scatter(x1_pos_test, x2_pos_test, marker="o", label="pass", color="
plt_neg_test = plt.scatter(x1_neg_test, x2_neg_test, marker="x", label="fail", color="
plt.legend((plt_pos_test,plt_neg_test),("admitted","not admitted"),loc='upper right')
plt.xlabel("Exam 1 Score $x_1$")
plt.ylabel("Exam 2 Score $x_2$")
plt.show()
```





1.1.3 d. Using SciKit Learn

interesting result, it's the same score/prediction than with my defective testing above.

```
In [11]: from sklearn.linear_model import SGDClassifier

         model = SGDClassifier(loss="hinge", penalty="l2", max_iter=1000, tol=1e-3)
         reg = model.fit(X, y)
         reg.score(X_test, y_test)
         reg.predict(X_test)

Out[11]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

I need to check the solution, I am missing something here...