Logistic regression

Additional e-learning lectures

(please choose one of them if you would like to have some deeper understanding)
A good introduction ... prof. Andrew NG (Stanford University).
Often recommended...
One in polish ...

Binary logistic regression

Instead of implementing the algorithm from scratch, you can start with this implementation of linear regression

matrix_mul_for_linear_reg.py

Please import all the required libaries, packages and database (

Dataset

):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import os
path = os.getcwd() + '/ex2data1.txt'
data = pd.read_csv(path, header=None, names=['Exam 1', 'Exam 2',
'Admitted'])
```

Ex. 0.: Please get yourself familiar with the data.

Goal: In data you can find two dependent variables (features) - `Exam 1` and `Exam 2`. Our goal is to determine whether a student has been admitted to the university on the basis of exam results or not (target - `Admitted`). The prediction/explanation Y takes two possible values/classes - we are dealing with a *binary classification*. A value of 1 means that the student was admitted and a value of 0 means that the student was not admitted.

Ex. 1.: Please follow the steps below:

- divide the data into parameters (X) and labels/classes (y)

```
X = data.values[:, :2].T
y = data.values[:, 2:].T
```

- create theta vector

```
theta = np.zeros((X.shape[0]+1, 1))
```

- add a row with values '1' on the top of the X matrix

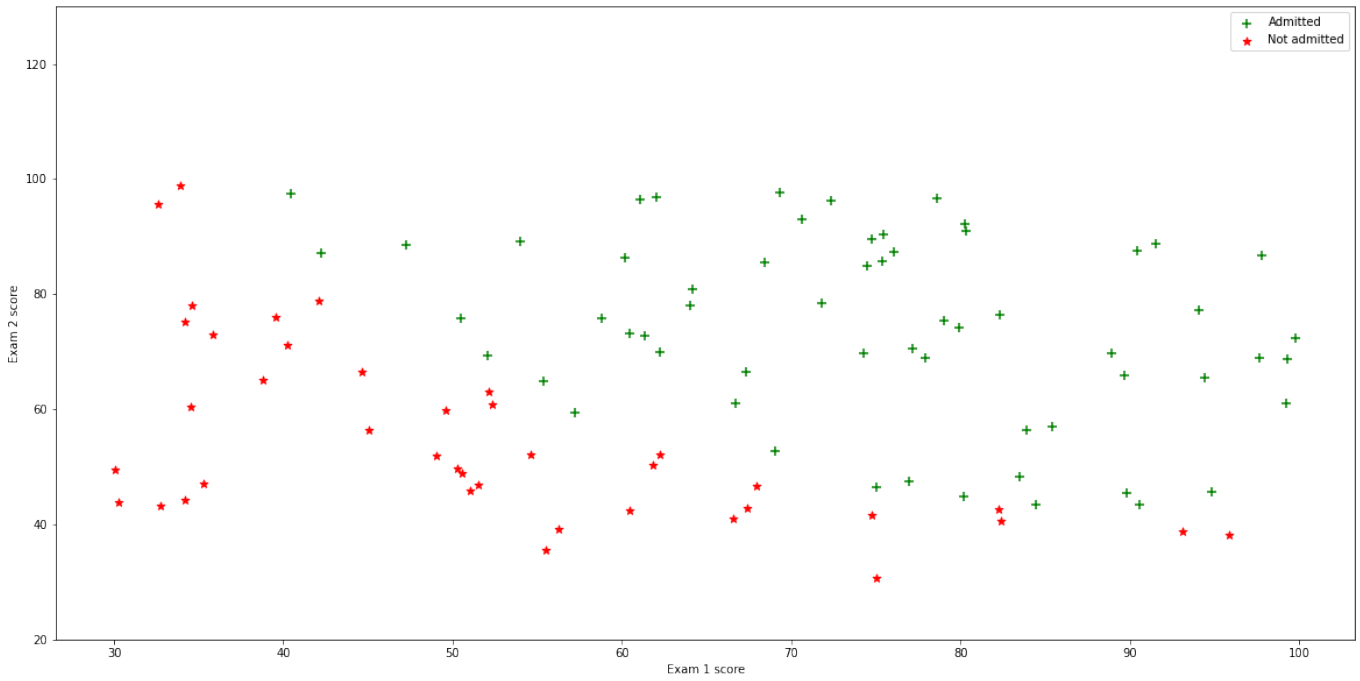
```
X = np.concatenate([np.ones([1, X.shape[1]]), X])
```

- normalize features of X matrix

```
for i in range(1, X.shape[0]):
    X[i, :] = (X[i, :] - np.mean(X[i, :])) / (np.std(X[i, :]))
```

Ex. 2.: Please plot the loaded data in the graph (using `scatter plot`). Can be both: a 2D plot with marks colored regarding to the class or a 3D plot with the class showed as 3rd dimension.

Example of a 2D plot:



Ex. 3.: Earlier, we said that we want our $h_\theta(x)$ classifier to match the property:

$$0 \leq h_\theta(x) \leq 1$$

In linear regression, we used the hypothesis:

$$h_\theta(x) = \theta^{T}x$$

Now we are making a slight modification, namely we are putting together the hypothesis function from linear regression with the new function sig:

$$h_\theta(x) = sig(\theta^{T}x)$$

Function sig it's so called logistic function (sigmoid):

$$sig(z) = \frac{1}{1+e^{-z}}$$

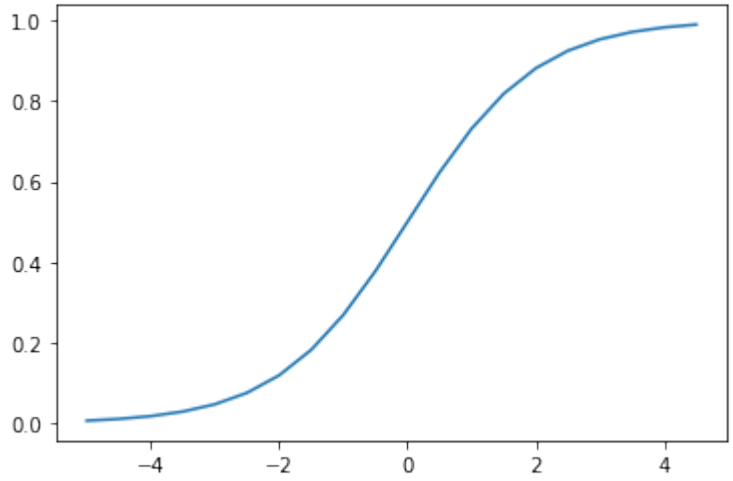
What boils down to the fact that:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^{T}x}}$$ Please implement a logistic function (sigmoid): $$sig(t)=\frac{1}{1+e^{-t}}$$

```
def sig(t)
#TODO
```

Ex. 4.: Using the function `np.arange` or `np.linspace`, please generate data from the range `[-5.5]`, step `0.5` and check the correct operation of the implemented function by plotting it.

Note that the sigmoid has an asymptote of $y=1$ going to $+\infty$, and of $y=0$ going to $-\infty$.



Cost function for logistic regression

We have a training set $ {(x_{1},y_{1}),(x_{2},y_{2}),\ldots,(x_{m},y_{m})} $, consisting of m training examples.

How to choose the θ parameters?

For linear regression, we used mean square error $$ J(\theta) = \frac{1}{m} \sum\limits_{i=1}^m \frac{1}{2}(h_\theta (x_{i})-y_{i})^2 $$

Let's define it as follows: $$ J(\theta) = \frac{1}{m} \sum\limits_{i=1}^m Cost(h_\theta (x_{i}), y_{i}) $$ $$ Cost(h_\theta (x),y) = \frac{1}{2}(h_\theta (x)-y)^2 $$

The $Cost$ function represents the cost that our algorithm must “pay” if it returns $h_\theta(x)$ when the actual value is y.

We can use this function when minimizing logistic regression, but there is a problem, it is a non-convex function. In the hypothesis, the sigmoid introduces a nonlinearity, which causes the formation of many local minima in the above cost function.

We will use the following cost function: $$ Cost(h_\theta (x),y) = \left\{\begin{matrix} -\log(h_\theta(x)) & \text{when } y=1\\ -\log(1-h_\theta(x)) & \text{when } y=0 \end{matrix}\right. $$

$$Cost(h_\theta(x), y) = -y log (h_\theta (x)) - (1-y)log(1 - h_\theta(x))$$ We can now substitute the above formula with $J(\theta)$ $$J(\theta) = \frac{1}{m} \sum^m_{i=1}Cost(h_\theta(x_{i}),y_{i}) = \frac{1}{m} \sum^m_{i=1} [-y_{i}log (h_\theta(x_{i})) - (1 - y_{i})log(1 - h_\theta(x_{i}))]$$

You may wonder why we chose this cost feature. Apart from the fact that it has the positive feature of being convex, it can be derived from the laws of statistics using the principle of maximum likelihood, i.e. the principle of how to efficiently search for the parameters of various models.

Ex. 5. Based on the above formulas, please implement the cost function J (recommended vectorized solution):

```
def cost(theta, X, y):
#TODO
```

To test the `cost` function, please initialize the θ values:

```
theta = np.zeros((X.shape[0], 1))
```

The correct value of the cost function for the analyzed data is `0.69`.

Simple gradient method

Our further goal is to find θ parameters for which $J(\theta)$ will be minimal. This can be achieved with the simple gradient method, which is to repeat $$\theta_j = \theta_j - \alpha \frac{d}{d\theta_j}J(\theta)$$ Where α is the step length (so called learning rate) in each iteration.

Derivative: $$\frac{d}{d\theta_j}J(\theta) = \frac{1}{m} \sum^m_{i=1}(h_\theta(x_{i})-y_{i})x_{ij}$$

Ex. 6.: Please implement simple gradient function:

```
def simple_gradient(X, y, theta, alpha, it):
    # it - number of iterations


    return theta, cost
```

For the parameter `alpha = 1` and 150 iterations, the cost function is around `0.20` and the values of θ [1.65947664], [3.8670477], [3.60347302]. Results may vary.

Ex. 7.: Please present the accuracy of the algorithm performance. Prediction values will be in the range [0; 1]. Threshold value = 0.5.

Ex. 8.: It is sometimes useful to be able to visualize the boundary line dividing the input space in which points are classified as belonging to the class of interest ($y=1$) from that space in which point do not ($y=0$).

This could be achieved by calculating the prediction associated with y for a mesh of (x_{1},x_{2}) points and plotting a 3D contour plot.

Alternatively, one can think of the decision boundary as the 2D line $x_{2}=mx_{1}+c$, being defined by points for which $y=0.5$ and hence $z=0$. For $x_{1}=0$ we have $x_{2}=c$ (the intercept) and assuming:

$$ z=\theta^{T}x+b $$

$$ z=ln(\frac{y}{1-y}) \Rightarrow y = \sigma(z) = \frac{1}{1+e^{-z}}$$

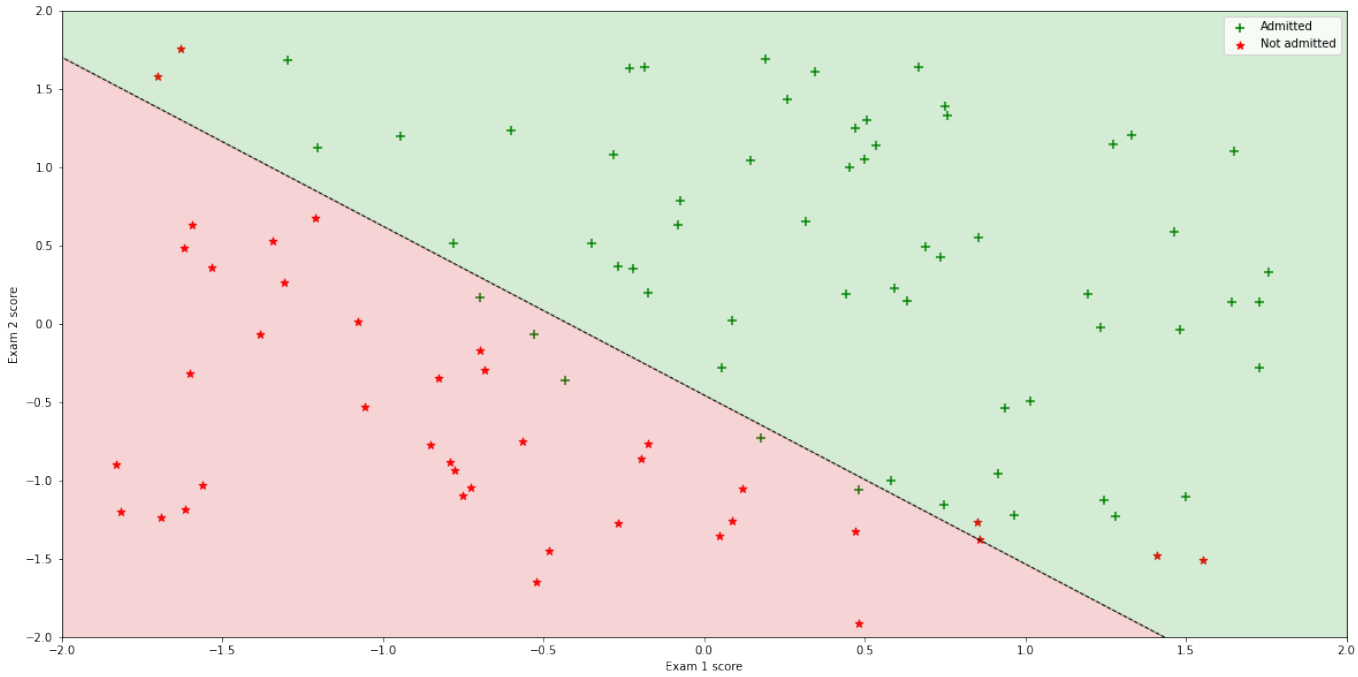
decision boundary line parameters are as follows:

$$ 0 = 0 + \theta_{2}x_{2} + b \Rightarrow c = - \frac{b}{\theta_{2}} $$

$$ m = -\frac{\theta_{1}}{\theta_{2}}$$

Please plot the decision boundary (as a line $x_{2}=mx_{1}+c$) on the data plot. If you have normalized your data you should use normalized data as background.

Expected output (no need to color/fill the area; the boundary is the crucial part):



One of the simplest approaches to plot decision boundary could be as follow:

```
x1 = np.arange(-2.0, 2.0, 0.1)
x2 = -theta[0, 0]/theta[2, 0] - theta[1, 0]/theta[2, 0] * x1
plt.figure()
plt.plot(x1, x2)
X1_1 = X[1, y[0, :] == 1.0]
X2_1 = X[2, y[0, :] == 1.0]
X1_0 = X[1, y[0, :] == 0.0]
X2_0 = X[2, y[0, :] == 0.0]
plt.plot(X1_1, X2_1, 'o')
plt.plot(X1_0, X2_0, 'x')
plt.show()
```

Logistic regression - Python

An alternate solution to a “from scratch” logistic regression implmentation is the `scikit-learn` lib

with its `LogisticRegression` classifier.

Multi-class logistic regression

Please import basic libaries/packets and “Iris” database:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data[:, :2]  # we are analyzing only 2 parameters
Y = iris.target
```

Ex. 1.: Get yourself familiar with the object sklearn.linear_model.LogisticRegression and choose appropriate optimalization algorithm, regularization coefficient and create a regression model.
Ex. 2.: Create an instance and fit the data using `fit` method.

Code below allows you to display the shape of decision regions:

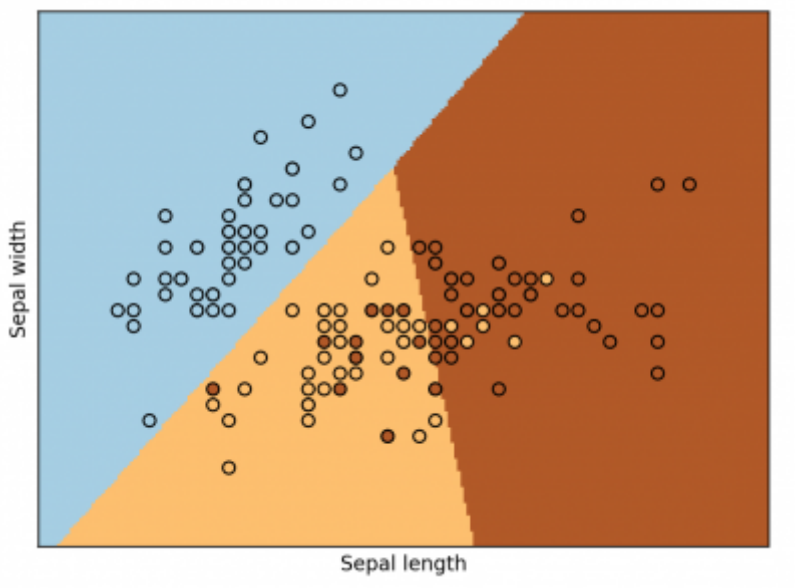
```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



Ex. 3.: Using method `predict_proba` please predict the probability of being a member of a given class.
Ex. 4.: Evaluation - please assess the accuracy of the algorithms using a variety of metrics (at least 2).