New Era University
College of Informatics and Computing Studies
Department of Computer Science

FREE ELECTIVE 3

CCSEL3-18

DeepLearning.AI

Supervised Machine Learning: Regression and Classification

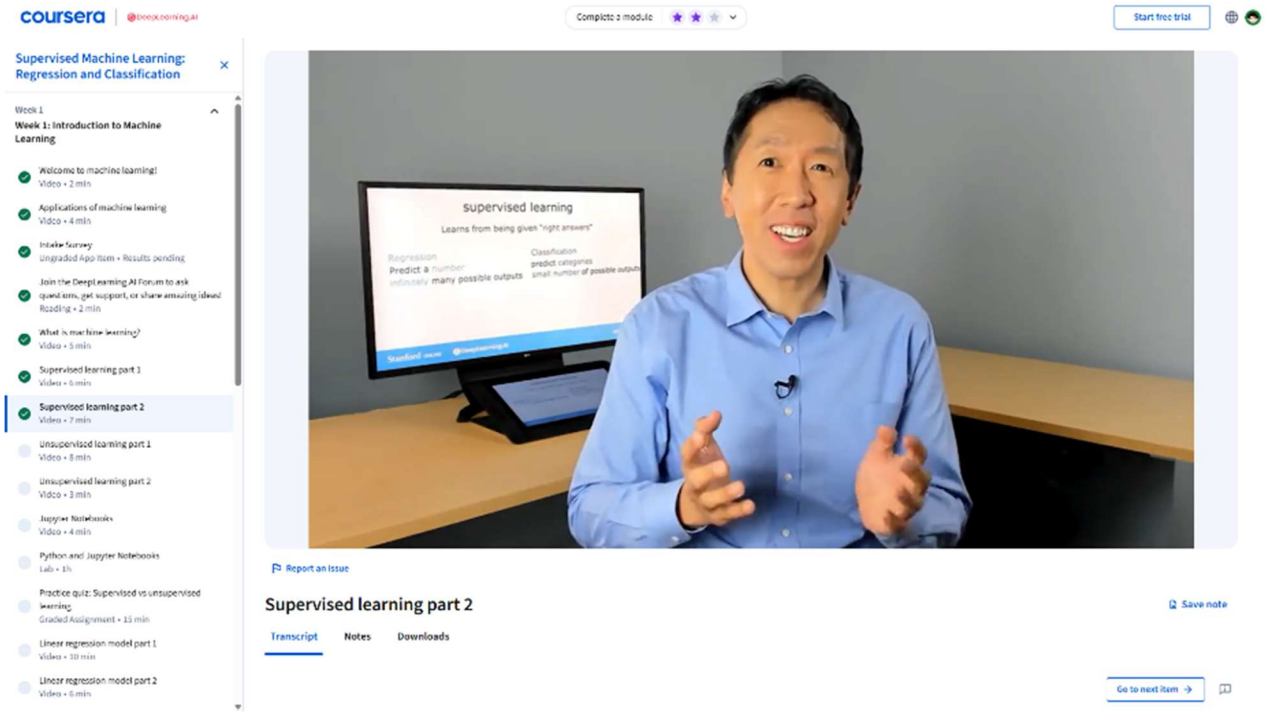
coursera

ONLINE COURSE DOCUMENTATION

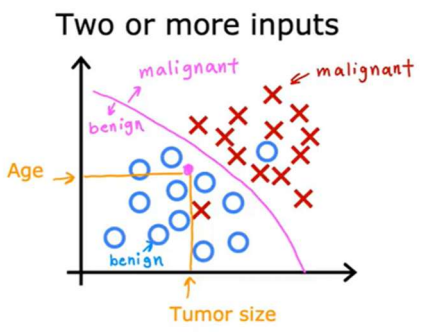
Marasigan, Vem Aiensi A.
BS Computer Science | 21-11295-310

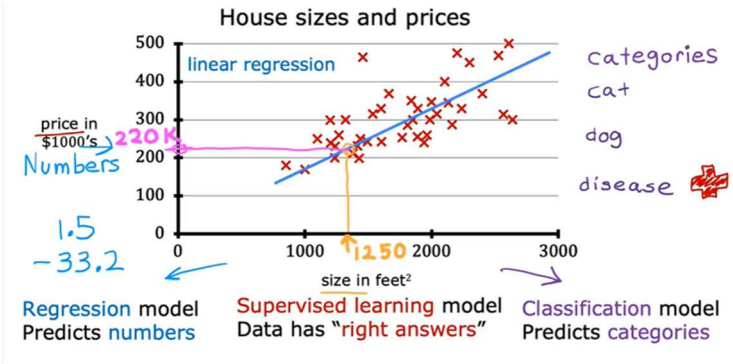
Module 1: Introduction to Machine Learning

[Completed by September 8 – 9]



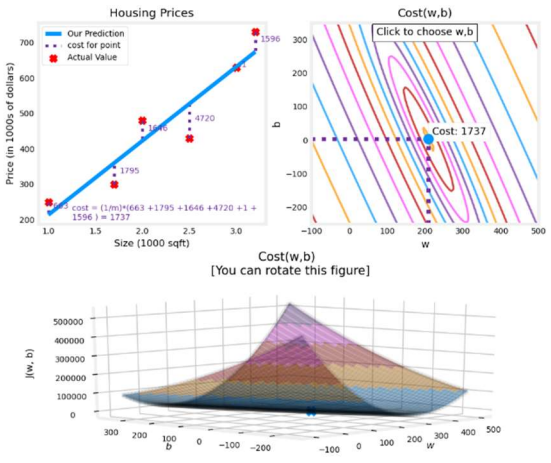
The first module/week for the course introduces machine learning concepts and foundational knowledge on how machine learning works as well as its applications. It showed different forms of machine learning which are; Regression, for predicting numeric values, and Classification, for predicting a category, and, how supervised learning differs from unsupervised learning.

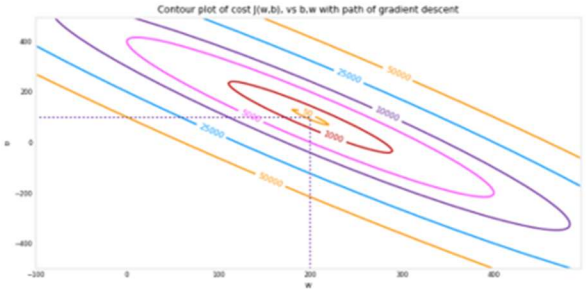




Proceeding, this module's topic is mostly around Regression, covering the most basic form of how predictions are achieved using Linear Regression Algorithm. This module discussed primary knowledge, from the linear function, which is also called the model, up to choosing the correct w

and b, and learning rate for the gradient descent function. This explained the functions needed in Linear Regression Algorithm which are: the model's f(x), the cost function that determines whether the w and b is good, and finally the gradient descent function which helps determine the best w and b that achieve the minimum value of cost function.





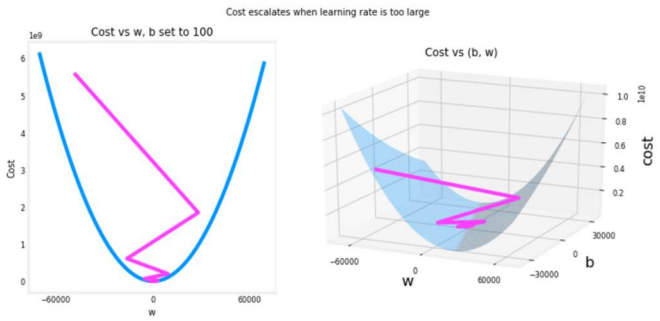
Also, learning all these functions was quite overwhelming at first. But with the aid of the laboratory Jupiter notebooks, with visualizations and interactivity, learning the concepts became much easier.

In [21]:
```
# initialize parameters
w_init = 0
b_init = 0
# set alpha to a large value
iterations = 10
tmp_alpha = 8.0e-1
# run gradient descent
w_final, b_final, J_hist, p_hist = gradient_descent(x_train ,y_train, w_init, b_init, tmp_alpha,
                                                    iterations, compute_cost, compute_gradient)
```

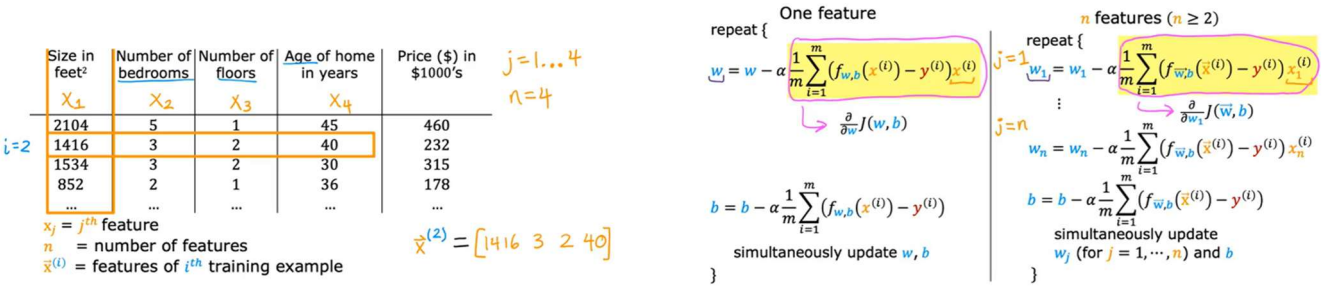
```
Iteration    0: Cost 2.58e+05  dj_dw: -6.500e+02, dj_db: -4.000e+02   w:  5.200e+02, b: 3.20000e+02
Iteration    1: Cost 7.82e+05  dj_dw:  1.130e+03, dj_db:  7.000e+02   w: -3.840e+02, b:-2.40000e+02
Iteration    2: Cost 2.37e+06  dj_dw: -1.970e+03, dj_db: -1.216e+03   w:  1.192e+03, b: 7.32800e+02
Iteration    3: Cost 7.19e+06  dj_dw:  3.429e+03, dj_db:  2.121e+03   w: -1.551e+03, b:-9.63840e+02
Iteration    4: Cost 2.18e+07  dj_dw: -5.974e+03, dj_db: -3.691e+03   w:  3.228e+03, b: 1.98886e+03
Iteration    5: Cost 6.62e+07  dj_dw:  1.040e+04, dj_db:  6.431e+03   w: -5.095e+03, b:-3.15579e+03
Iteration    6: Cost 2.01e+08  dj_dw: -1.812e+04, dj_db: -1.120e+04   w:  9.402e+03, b: 5.80237e+03
Iteration    7: Cost 6.09e+08  dj_dw:  3.156e+04, dj_db:  1.950e+04   w: -1.584e+04, b:-9.80139e+03
Iteration    8: Cost 1.85e+09  dj_dw: -5.496e+04, dj_db: -3.397e+04   w:  2.813e+04, b: 1.73730e+04
Iteration    9: Cost 5.60e+09  dj_dw:  9.572e+04, dj_db:  5.916e+04   w: -4.845e+04, b:-2.99567e+04
```

Above, w and b are bouncing back and forth between positive and negative with the absolute value increasing with each iteration. Further, each iteration $\frac{\partial J(w,b)}{\partial w}$ changes sign and cost is increasing rather than decreasing. This is a clear sign that the *learning rate is too large* and the solution is diverging. Let's visualize this with a plot.

In [22]:
```
plt_divergence(p_hist, J_hist,x_train, y_train)
plt.show()
```



Module 2: Regression with multiple input variables [Completed by September 16]



Taking a step further, learning multiple linear regression involved a lot of concepts. Considering multiple features, optimizations are introduced in this module. These are **vectorization**, **feature scaling**, **feature engineering**, and a little bit of polynomial regression example.

Vectorization

Due to the nature of samples having multiple features (x_1, x_2, x_3 ... x_n), the resulting model would be:

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

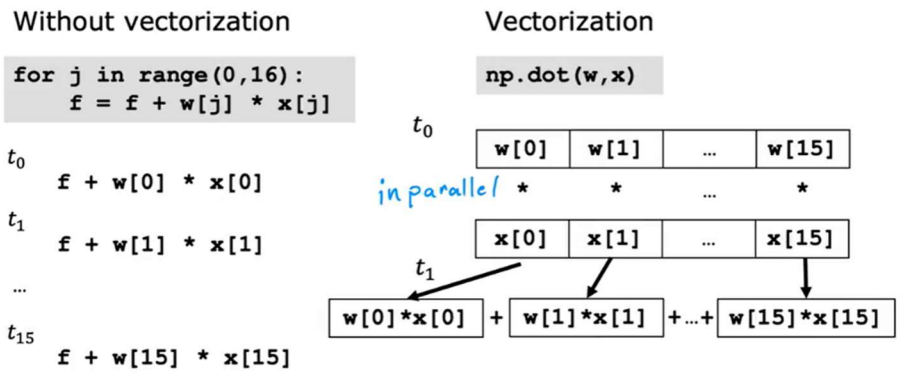
which can range up to thousands of features. Say 1000 features, this approach would be inefficient considering that it must iterate in every cost function.

Hence, this is where Vectorization comes in by using functions for sets, on of which is the dot product function. This can be achieved in code as well using the NumPy library's dot function

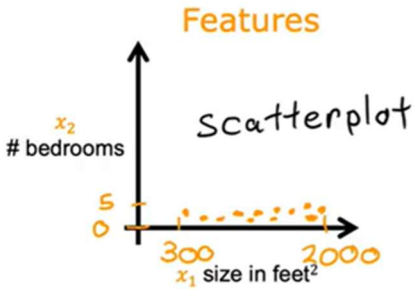
$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

```
f = np.dot(w,x) + b
```

This makes the formula representation cleaner and especially execute the code much faster.

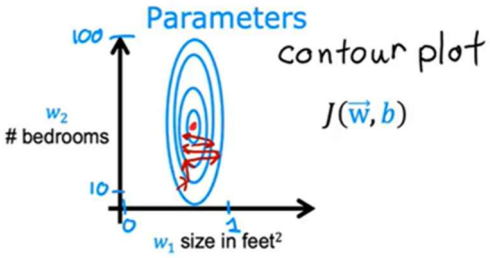


Feature Scaling



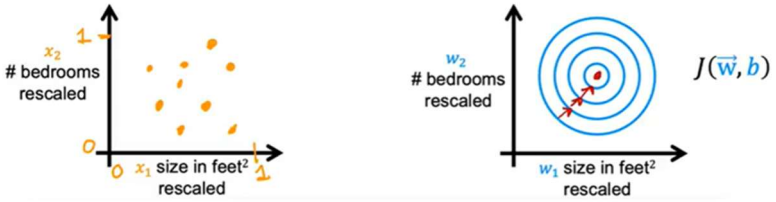
Another thing to consider in multiple linear regression would be the range of values for each feature. Having a large number in range in other features affects how gradient descent works to achieve minimum. Say:

$$300 \leq x_1 \leq 2000 \qquad 0 \leq x_2 \leq 5$$



If the difference in the ranges is far from each other, these might introduce inefficiency or overshooting when performing gradient descent to reach minimum.

So, the thing to do here is to transform feature values so that features can be comparable to other features' range of values. This is called feature scaling, can be normalization as well, and when features are rescaled first, this would be the gradient descent:



There are many ways to rescale features, these are:

Basic Feature scaling

$$x_{1,scaled} = \frac{x_1}{2000}$$

Mean normalization

$$x_1 = \frac{x_1 - \mu_1}{2000-300}$$

max - min

Z-score normalization

$$x_1 = \frac{x_1 - \mu_1}{\sigma_1}$$

Rescaling all features not only makes a better representation of a graph but most importantly help gradient descent find the minimum costs for each w_s considering multiple features.

Feature Engineering

Having multiple features to train a model, another we can do is to add custom features based on existing ones. These relies on our intuition whether if some features are combined can have significance for predictions.

As a basic example from the course, say existing features are:

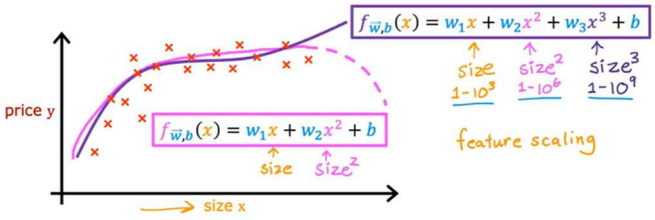
x_1 is frontage and,
x_2 is depth

$$area = frontage \times depth$$

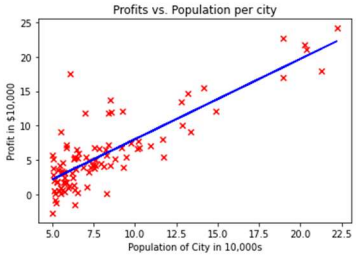
Thinking that the area of the lot has a significance, say where predicting its price, we may create another feature "area" to be our x_3

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

By the end of the module, we get a glimpse of **Polynomial Regression**. This is used for scenarios where there is a tragectory where a curve is much more suitable instead of a straight line, linear regression.




```
In [27]: # Plot the linear fit
         plt.plot(x_train, predicted, c = "b")

         # Create a scatter plot of the data.
         plt.scatter(x_train, y_train, marker='x', c='r')

         # Set the title
         plt.title("Profits vs. Population per city")
         # Set the y-axis label
         plt.ylabel('Profit in $10,000')
         # Set the x-axis label
         plt.xlabel('Population of City in 10,000s')

Out[27]: Text(0.5, 0, 'Population of City in 10,000s')
```



And finally, a laboratory activity practicing what was learned so far in linear regression where we implement functions in code.

```
# You need to return the following variables correctly
dj_dw = 0
dj_db = 0

### START CODE HERE ###
for i in range(m):
    dj_dw += (w * x[i] + b - y[i]) * x[i]
    dj_db += w * x[i] + b - y[i]

dj_dw /= m
dj_db /= m
```

Module 3: Classification

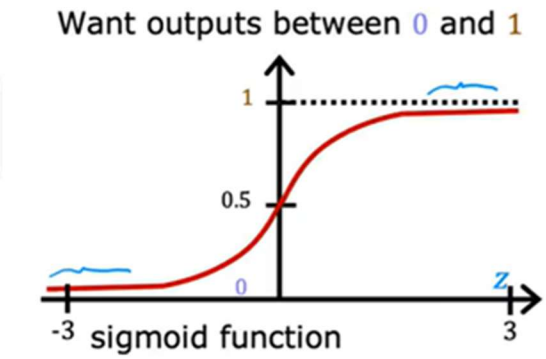
[Completed by September 17]

From the previous lessons about Linear Regression for predicting numerical values, this module now moves on to predicting classes or categories. This is done through Logistic Regression. Although it is slightly different from linear regression, most concepts and knowledge from previous lessons do carry-over in Logistic Regression, from computing gradient descent, rescaling, and even feature engineering. Still, these are the key differences:

Logistic Function model, or the **Sigmoid** function

$$f_{\vec{w},b}(\vec{x}) = g(\underbrace{\vec{w}\cdot\vec{x}+b}_{z}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$$

Considering classification, having only 2 values, either 0 or 1, true or false, or in short binary outputs, a linear regression model won't be reliable. Hence, we can use the sigmoid function that produces outputs ranging from 0 to 1, then setting up a threshold or decision boundary to determine if the prediction is positive, or if it is negative.



Also, having a new model also changes the cost function used in the gradient descent. This incorporates another function which is called the **Loss function**

$$\begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=1 \\ -\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)}=0 \end{cases}$$

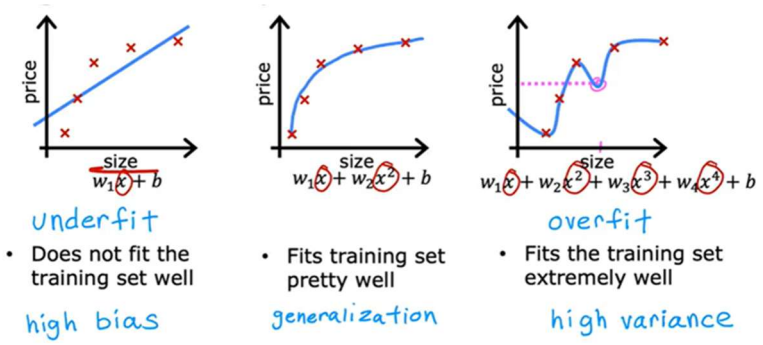
convex
↳ can reach a global minimum

And for simplicity, for programming as well, this can also be written as:

$$y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

Decision Boundary – This is the main objective of using logistic regression, this decision boundary is the resulting graph of the logistic regression model that will determine the category of the sample being predicted.

Another concept introduced in this module was **Generalization,** which applies to both Linear and Logistic Regression. This involves the decision of on what degree should the model be to create a "just right" curve of predictions for a particular dataset. This generalization is done to avoid underfitting and overfitting and is achieved by regularization. This procedure involves many ways which are:

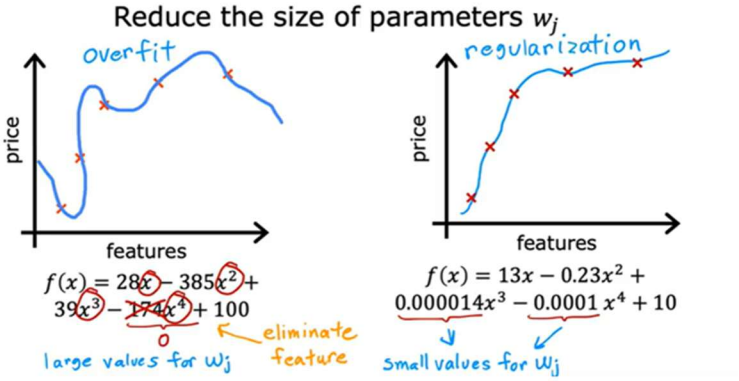


Eliminating other features, features that seems to be "not that significant". This way, the model can reduce being overfitted. However, what if all features do mean significant factor for prediction, this is where regularization comes in.

Regularization is done by reducing the values of the parameters, this is the ***w*** for every feature x. This is done by incorporating the equation:

$$\frac{\lambda}{2m}\sum_{j=0}^{n-1} w_j^2$$

With this the resulting formula for cost function and gradient descent would be respectively:



Cost Function:

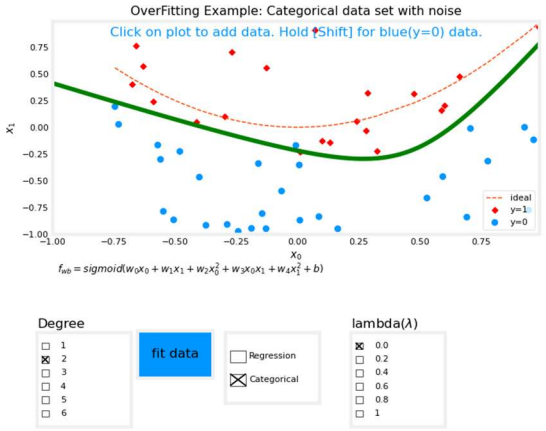
$$J(\mathbf{w}, b) = \frac{1}{m}\sum_{i=0}^{m-1}\left[-y^{(i)}\log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=0}^{n-1} w_j^2$$

Gradient Descent:

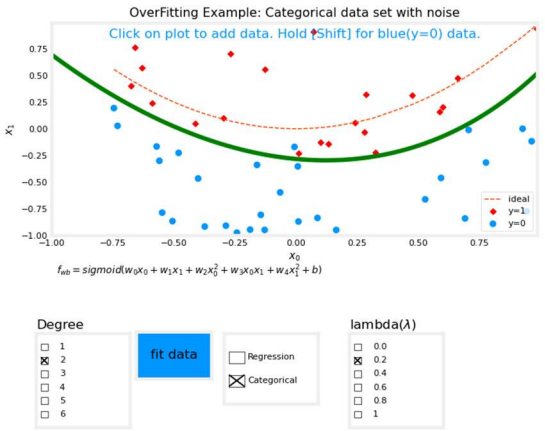
$$\frac{\partial J(\mathbf{w}, b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}\left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}\right)x_j^{(i)} + \frac{\lambda}{m}w_j$$

$$\frac{\partial J(\mathbf{w}, b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}\left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)}\right)$$

The key variable in this regularization is the **lambda**. Setting small values for lambda will reduce the features' influence in the model, and vise-versa, effectively reducing overfitting when a "just right value is used" achieving a generalized model. Here's a comparison below from the module:



Without Regularization

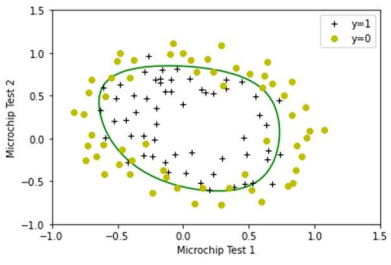


With Regularization

This module also involved a practice lab to implement Logistic Regression. Here's a small snapshot of my take:

```
In [41]: # UNQ_C6
def compute_gradient_reg(X, y, w, b, lambda_ = 1):
    """
    Computes the gradient for logistic regression with regularization

    Args:
      X : (ndarray Shape (m,n)) data, m examples by n features
      y : (ndarray Shape (m,))  target value
      w : (ndarray Shape (n,))  values of parameters of the model
      b : (scalar)              value of bias parameter of the model
      lambda_ : (scalar,float)  regularization constant
    Returns
      dj_db : (scalar)             The gradient of the cost w.r.t. the parameter b.
      dj_dw : (ndarray Shape (n,)) The gradient of the cost w.r.t. the parameters w.

    """
    m, n = X.shape

    dj_db, dj_dw = compute_gradient(X, y, w, b)

    ### START CODE HERE ###
    for i in range(n):
        reg = w[i] * (lambda_ / m)
        dj_dw[i] += reg
    ### END CODE HERE ###

    return dj_db, dj_dw
```

```
In [44]: plot_decision_boundary(w, b, X_mapped, y_train)
# Set the y-axis label
plt.ylabel('Microchip Test 2')
# Set the x-axis label
plt.xlabel('Microchip Test 1')
plt.legend(loc="upper right")
plt.show()
```



Note: All these laboratory activities, as well as the optional labs, are also uploaded.
You may check that out as well by scanning the QR code at the front page.

COMPLETION RECORD

Machine Learning

Supervised Machine Learning: Regression and Classification

Course 1 of 3 · Complete

You can access your certificate on September 23, 2025 when your free trial ends.

Get your certificate early