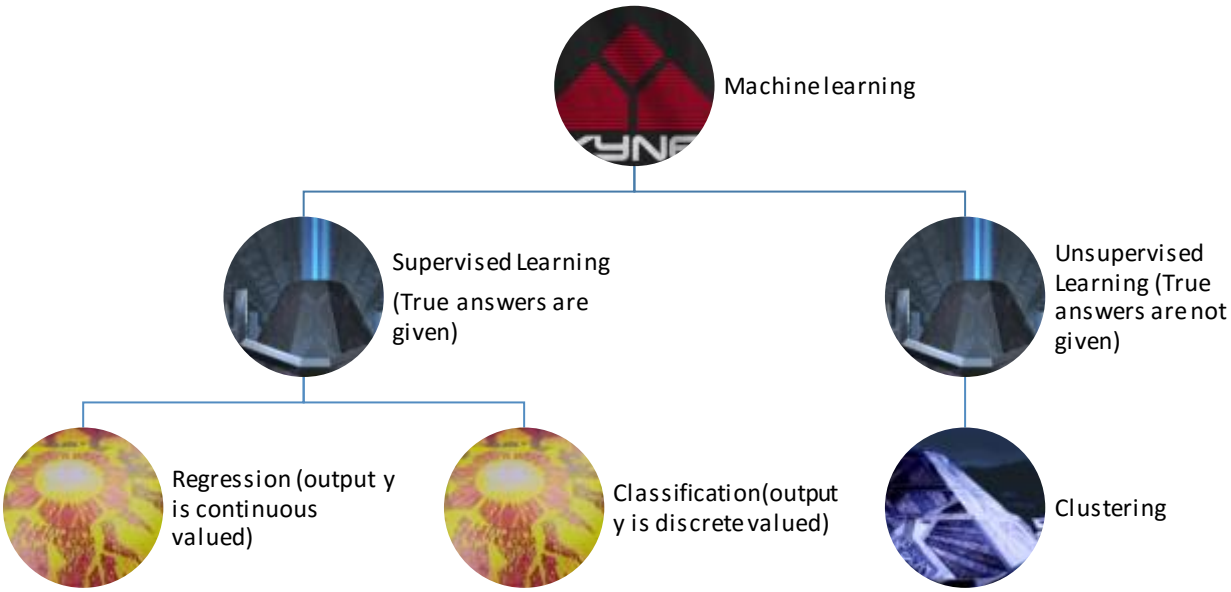
A WALKTHROUGH TO SKYNET
By Elgiz Baskaya

This is a **step by step** guide to implement machine learning to various problems. But beware that **is not done by an expert but just a fresh beginner** to the challenging world of capable machines.

The aim of machine learning methods is to train a model with a given data set in order to predict the output values of a new input. There are a lot of more formal or may be more generalized definitions which I find them not very easy to internalize.

The methods are mainly handled under two main categories: the supervised and unsupervised learning.



And beware that how humans are still in the loop.. A long walk until Skynet actually.

HUMAN	MACHINE
Select **Output y** (for regression), **Classes** (for classification)	**Cluster data** with respect to Some chosen criteria?
Select **Features** (the inputs of the problem or you can say the states that effect the output)	
Select	**Select**

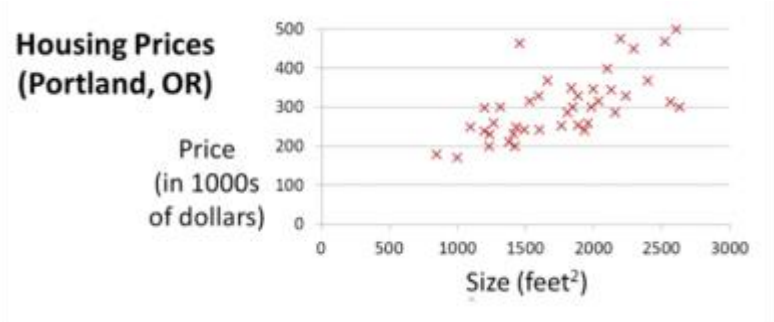
The structure of the hypothesis function Ex: $h(x) = \theta_0 + \theta_1 x$	**Parameters** of the hypothesis function Ex : for the hypothesis given left; θ_0, θ_1
Select **The structure** of the cost function	**Select** **Parameters** of the cost function

1. VISUALIZING THE DATA

To have an understanding of data makes gives you an idea how to tackle the problem. At the end you are the one selecting the output, features, the hypothesis, cost function. It is true that the algorithms are optimizing some parameters of the hypothesis and the cost function but still the structure itself is supplied by a human. So yes, machines have a way to go..

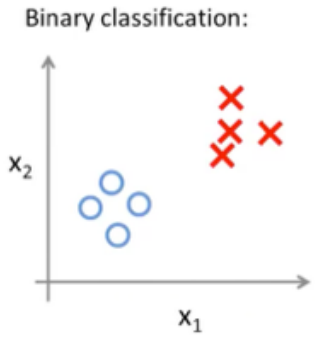
Linear Regression-> Easy to see the structure with one feature

Example:



This example is taken from Machine Learning class of Coursera by Andrew Ng. In this example y (target / output variable) is the price of the house and size is the feature (input variable). This is a one feature problem which seems to be modeled by a linear hypothesis. So you can apply linear regression.

Logistic Regression-> This is a classification problem since the aim is to distinguish the output value y belongs to a class. Here our vertical dimension is not the output as the previous example of linear regression but it is another feature (x2 in this example). And the information about the output is stored in the signs (O and X)



Here in this example it seems that you can apply logistic regression since the data seems to be separated by a linear *decision boundary*.

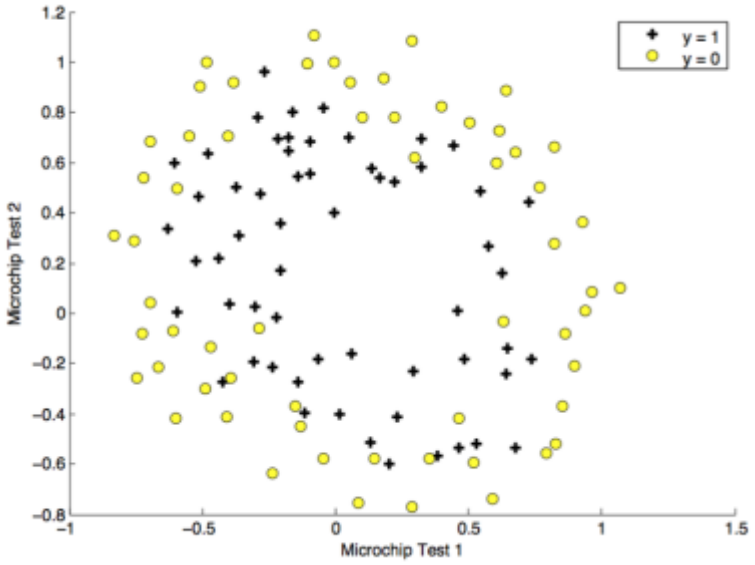
How to have that sense if there are more features (biiippp)??

2. FEATURE MAPPING

Depending on the results from visualizing data, you might need to map your features. This is because the straightforward implementation of linear / logistic regression will end up in linear model, linear decision boundary respectively. And if you realize that your model or decision boundary being linear will not be enough to satisfactorily describe your training data, you might map your features.

$$
\begin{aligned}
h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\
&= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \\
x_1 &= (size) \\
x_2 &= (size)^2 \\
x_3 &= (size)^3
\end{aligned}
$$

Below you can see a data set that will be used for logistic regression for the purpose of classification of a microchip assessment. It is seen from the plot that a linear decision boundary will not serve appropriately. So we might map our features as below:



$$
\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}
$$

But keep in mind that with a higher dimensional feature vector, *overfitting* might become a problem. Which can be handled via *regularization(Step 6)*.

3. SCALING

The next step is to normalize the features of the data to make the values of features change with the same order of magnitude. The reason is about the calculation of parameters of the hypothesis via an optimization algorithm and its convergence rate. Although there are different methods for normalization, a common one is to manage it

$$x_j = \frac{x_j - \mu_j}{s_j}$$

$$\mu_j = \frac{\sum_{i=1}^{m} x_j^i}{m}$$

$$s_j = \max(x_j) - \min(x_j)$$

An important point is to keep that values (μ_j, s_j) and may be standard deviation if it is used instead of s_j. If you have already added artificial feature $x_0 = 1$ (step 4), do not apply scaling to the artificial feature.

4. ADD ARTIFICIAL FEATURE $x_0 = 1$

This is for the purposes of generalizing the hypothesis for more features and the ability to write it in vectorized form.

Such that a multiple feature

$$\vec{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \quad x_0 = 1 \quad \text{parameter vector} \equiv \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix}$$

since the hypothesis for linear regression is

$h_\theta(x) = \theta_0 + \theta_1 x_1$ for one feature

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_N x_N$ for multiple features where N is the number of features

can be written compactly as

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

JUST A REMINDER FOR INDEXING
Through this walkthrough

$$M \text{ is the number of training set and } i \text{ is the training example index}$$
$$i = 1 \cdots M$$

$$N \text{ is the number of features and } j \text{ is the feature index}$$
$$j = 1 \cdots N$$

So the X matrix will look like

$$\begin{bmatrix} 1 & f & f & \cdots & f \\ 1 & e & e & \cdots & e \\ 1 & a & q & \cdots & a \\ 1 & t & t & \cdots & t \\ 1 & u & u & \cdots & u \\ 1 & r & r & \cdots & r \\ 1 & e & e & \cdots & e \\ 1 & 1 & 2 & \cdots & N \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & \vdots & \vdots & \ddots & \vdots \end{bmatrix}_{M \times N} \begin{matrix} \rightarrow first\ training\ example \\ \rightarrow second\ training\ example \\ \rightarrow third\ traning\ example \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \rightarrow M\ th\ training\ example \end{matrix}$$

5. SELECTION OF COST FUNCTION AND CALCULATE GRADIENT

Cost function and the gradient are usually the inputs of the optimization algorithms by which you will calculate the optimized θ values to fit a model or a decision boundary by using the data set given to you. The selection of cost function is another issue but for standard applications there are widely used cost functions for each type of machine learning (e.g linear regression, logistic regression)

Cost function for Linear Regression

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Cost function for Logistic Regression (Classification) – a subcase of two class. $y \in \{0,1\}$

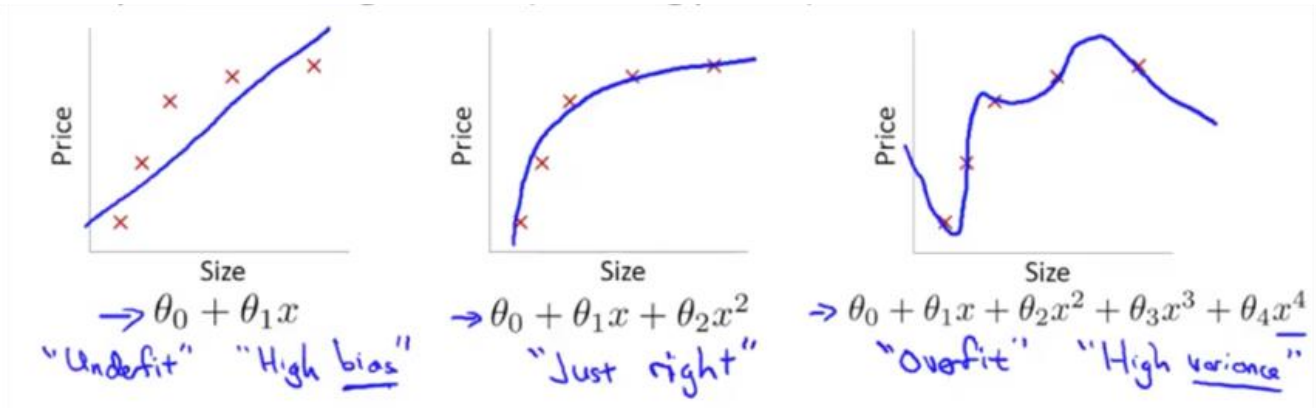
$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

6. IF NEEDED, MODIFY THE COST FUNCTION BY ADDING REGULARIZATION TERM

Regularization is done to avoid overfitting which can be roughly explained that it is when you end up with a very complicated model or decision boundary, especially when you have nonlinear feature terms such as below;

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

Overfitting might end up by fitting very well with the training set, but fail to generalize to the new data for the prediction phase.



You add a term to cost function to penalize parameters $\vec{\theta}$ by being large (i.e to make $\vec{\theta}$ as small as possible), but keep in mind that θ_0 is not penalized (by convention).

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

Here, the art is to find the appropriate value for lambda which serves like a weight between the usual part of the cost function and the second part, which penalize theta vector by being big. So it decreases the values of theta. But if this lambda is bigger than it supposed to be then you end up with

$$h_\theta(x) = \theta_0$$

since you penalized all terms except θ_0 (as we earlier said, θ_0 is not penalized by convention) Such that:

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Thus you have an underfit to the model such as

7. COST FUNCTION MINIMIZATION

Here is the point to use the optimization algorithms to minimize cost function by changing the parameters $\vec{\theta}$ (i.e not by changing x and y)

$$\text{Goal: } \underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

For that there are different optimization algorithms. One of them is the Gradient Descent. Others are Conjugate Gradient, BFGS, L-BFGS. Most of them needs the cost function and its gradient to minimize the cost function. The selection of the optimization algorithm is another subject so for now you can select Gradient Descent.

Gradient Descent

For two features:

Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}

Linear Regression Model

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Gradient descent algorithm

repeat until convergence {

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ (for $j = 0$ and $j = 1$)

}

Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

if we calculate the gradient (partial derivative term) and substitute,

Gradient descent algorithm

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}$$

}

Beware that you have to simultaneously update the thetas, meaning that you need to evaluate hypothesis in the summation just above with the previous theta values. So during one iteration, you should use the same theta values to substitute into $h_\theta(x)$, in order to calculate next values of θ_0, θ_1.

For Multiple Features:

[theta_optimized, J_history] = gradientDescent(X, y, theta_initial, alpha, number_iteration)

inputs

X – Feature matrix (input matrix)
y – output vector (target variable vector)
theta_initial – initial guess for the optimized parameter theta (can start with zeros)
alpha – learning rate
number_iteration – number of iterations to find the optimized theta

outputs

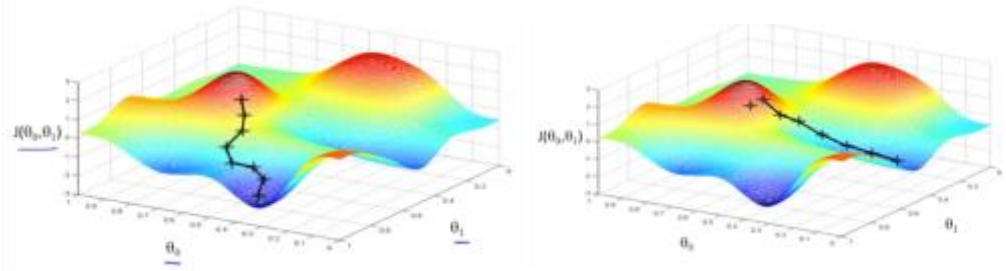
theta_optimized – theta vector that minimizes the cost function
J_history – the values of the cost function calculated during the course of iterations.

So a couple of ideas:

Choosing theta_initial:

- A conventional approach is to assign theta_initial = 0
  A point to know is that for different initial values your theta might converge to a different value (local minima but not the global one). But for linear regression it is shown that the cost function that we have selected in the notes above is convex, meaning that it has no local minima. It has only one global minima.

  An example of different initial conditions will converge to different solutions (slightly different choice of initial theta might make you converge to the local minima, which gives you J(theta) values smaller than some parts but does not give you the smallest value of J(theta))



Choosing alpha:

- Alpha, learning rate gives the gradient descent its step size, so a bigger value means a faster convergence. But keep in mind for a big alpha, your solution might not converge or even diverge. And too small value might end up the optimization to converge very slowly.
  So you can try in order these values of alpha = 0.001, 0.01, 0.1, 0.003, 0.03, 0.3

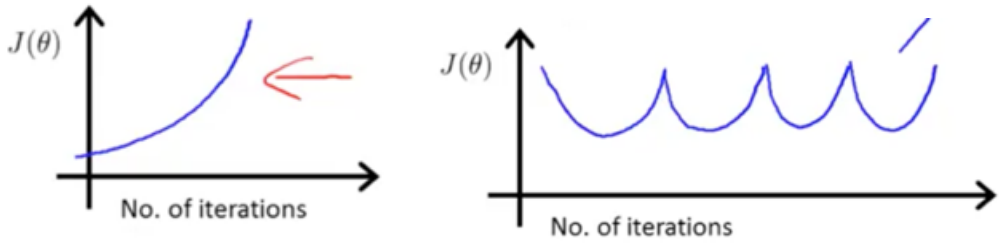
Choosing number_iterations:

You can star with 1000 and then by visualizing the cost function (Step 8) check if the cost function converged to a steady value or not. If it has not yet converged but decreasing, try a bigger value for number_iterations.

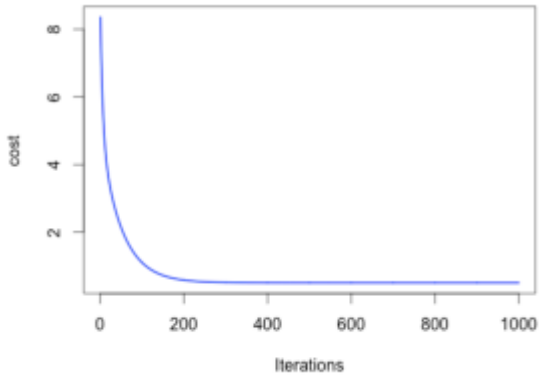
8. VISUALIZING COST FUNCTION

Check converge by visualizing the cost function as a function of iterations of the optimization algorithm. Normally you expect this plot to be decreasing, never increase, converging to a steady value at the end of the optimization algorithm.

To be able to observe how the cost function J(theta) during the iterative process of optimization, you should save the cost function values at each iteration.
Two examples of a bad optimization implementation



Not a good sign! Decrease alpha(learning rate)
What you expect is something like this



9. VISUALIZE THE MODEL/CLASSIFIER (DECISION BOUNDARY) WITH RESPECT TO TRAINING DATA.

Overfitting can be roughly explained that it is when you end up with a very complicated model or decision boundary. This ends up by fitting very well with the training set, but fail to generalize to the new data for the prediction phase. If overfitting occurs apply regularization meaning; Go step 6.

10. PREDICTION

For a new set of X(input), now you will predict the outcome thanks to the model/classifier you trained. Now you should first normalize the new data X_new by using the same mean and standard deviation values we had previously calculated from the training set in Step 2. And then evaluate h(theta) by the optimized theta and regularized new data set X_new_regul.

References

1. Andrew Ng, Machine Learning - Coursera