ML & Neural Network Tutorial

ANAIS 2023

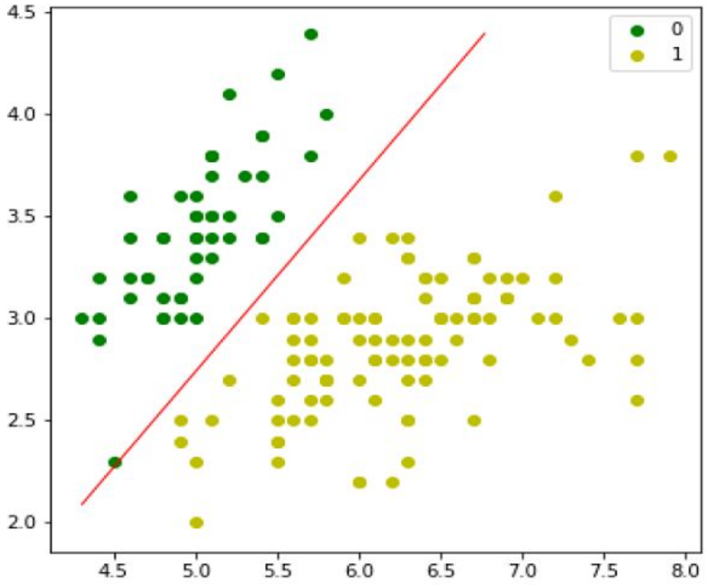
Nripesh Parajuli, PhD

Different types of ML algorithms

- KNNs
- Logistic regression
- Support vector machines (SVC / SVR)
- Random Forest
- Boosted trees
- Neural networks / Deep Learning

Linear model: Logistic regression

$$p(x) = \frac{1}{1 + e^{-\beta x}}$$

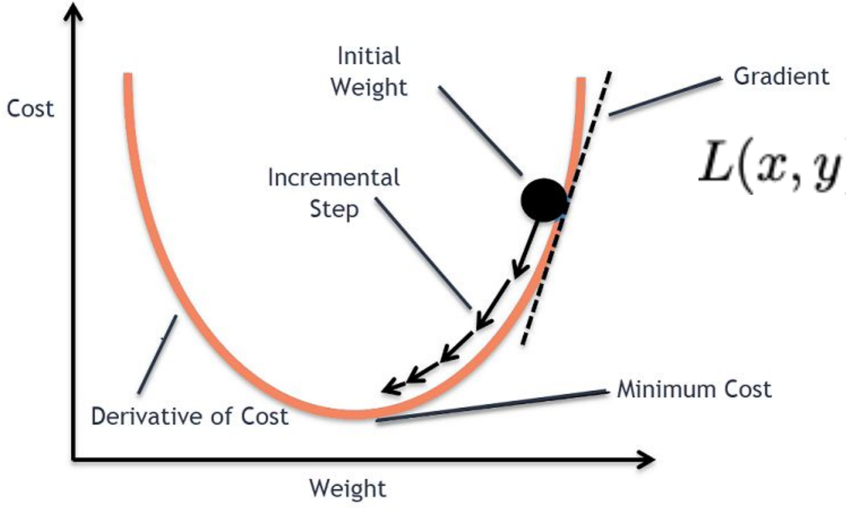


A line (in 2D) separates the 2 sets of points in a binary classification problem.

https://www.tutorialspoint.com/machine_learning_with_python/classification_algorithms_logistic_regression.htm

Finding the optimal parameters

- How to find the optimal beta?
- For 1-2 dimensions, we can do trial and error, but not scalable.
- We minimize the following loss function using gradient descent:



$$L(x,y) = -y * log(p(x)) - (1-y) * log(1-p(x))$$

$$\nabla_\beta L = \frac{dL(x,y;\beta)}{d\beta}$$

$$\beta' = \beta - LR * \nabla_\beta L$$

https://www.analyticsvidhya.com/blog/2020/10/how-does-the-gradient-descent-algorithm-work-in-machine-learning/

Gradient computation*

$$p(x) = \frac{1}{1+e^{-\beta x}}$$

$$L(x,y;\beta) = -yln(p(x)) - (1-y) * ln(1-p(x))$$

$$\frac{dL(x,y;\beta)}{d\beta} = \frac{dL(x,y;\beta)}{dp(x)} * \frac{dp(x)}{d\beta}$$

$$\begin{aligned}
\frac{dL(x,y;\beta)}{dp(x)} &= -\frac{d}{dp(x)}(yln(p(x)) - (1-y) * ln(1-p(x))) \\
&= -\frac{y}{p(x)} + \frac{1-y}{1-p(x)} \\
&= -\frac{y(1-p(x)) + (y-1)p(x)}{p(x)(1-p(x))} \\
&= -\frac{y-p(x)}{p(x)(1-p(x))}
\end{aligned}$$

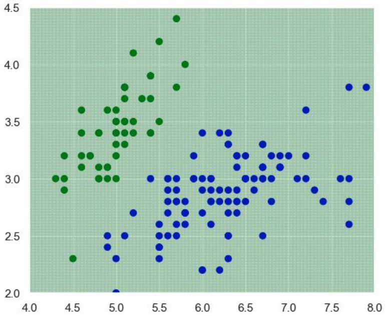
$$\begin{aligned}
\frac{dp(x)}{d\beta} &= \frac{d}{d(1+e^{-\beta x})}(1+e^{-\beta x})^{-1} * \frac{d(1+e^{-\beta x})}{d\beta} \\
&= -\frac{1}{(1+e^{-\beta x})^2}(-xe^{-\beta x}) \\
&= p(x)^2 * \frac{x(1-p(x))}{p(x)} = p(x)x(1-p(x))
\end{aligned}$$

$$\begin{aligned}
\frac{dL(x,y;\beta)}{d\beta} &= p(x)x(1-p(x)) * -\frac{(y-p(x))}{p(x)(1-p(x))} \\
&= -x(y-p(x)) \\
&= x(p(x)-y)
\end{aligned}$$

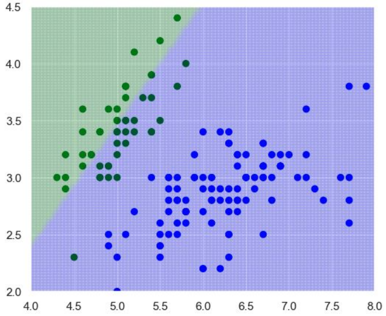
Gradient descent

- Initialize **beta** with some random values or **zero.**
- For each step:
  - Compute the gradient of the loss function at that value of beta, call it **grad.**
  - Update **beta = beta - LR * grad.** Where **LR** is the learning.
  - Terminate if the magnitude of the change (grad) is too small or after **T** number of iterations.
- Learning rate **(LR)** can be adapted as steps progress. Higher in the beginning and slower later.
- **T** also needs to be adapted depending on the problem.
- Gradient can be computed directly (exactly) or analytically using a small perturbation around the variable.

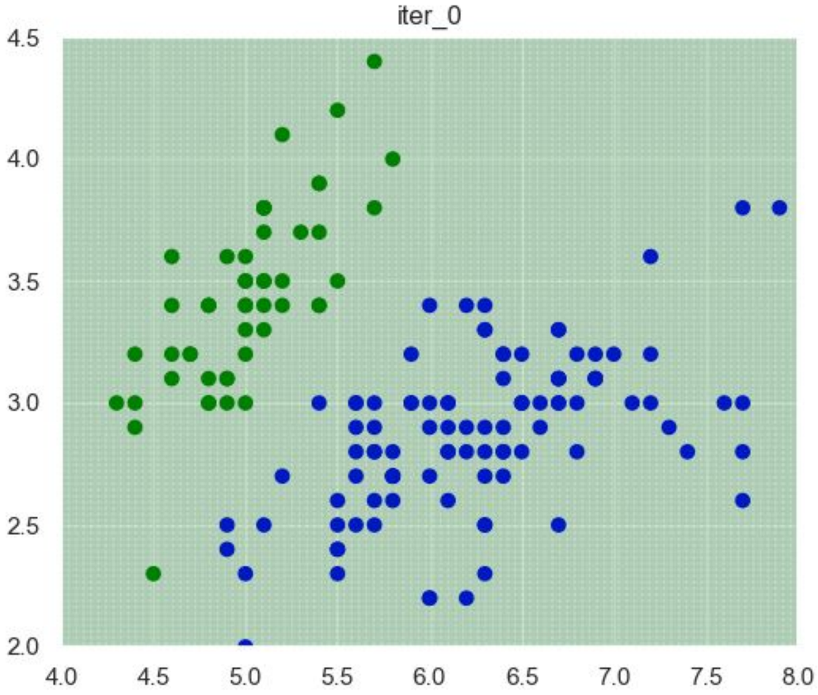
Gradient descent



iter=0

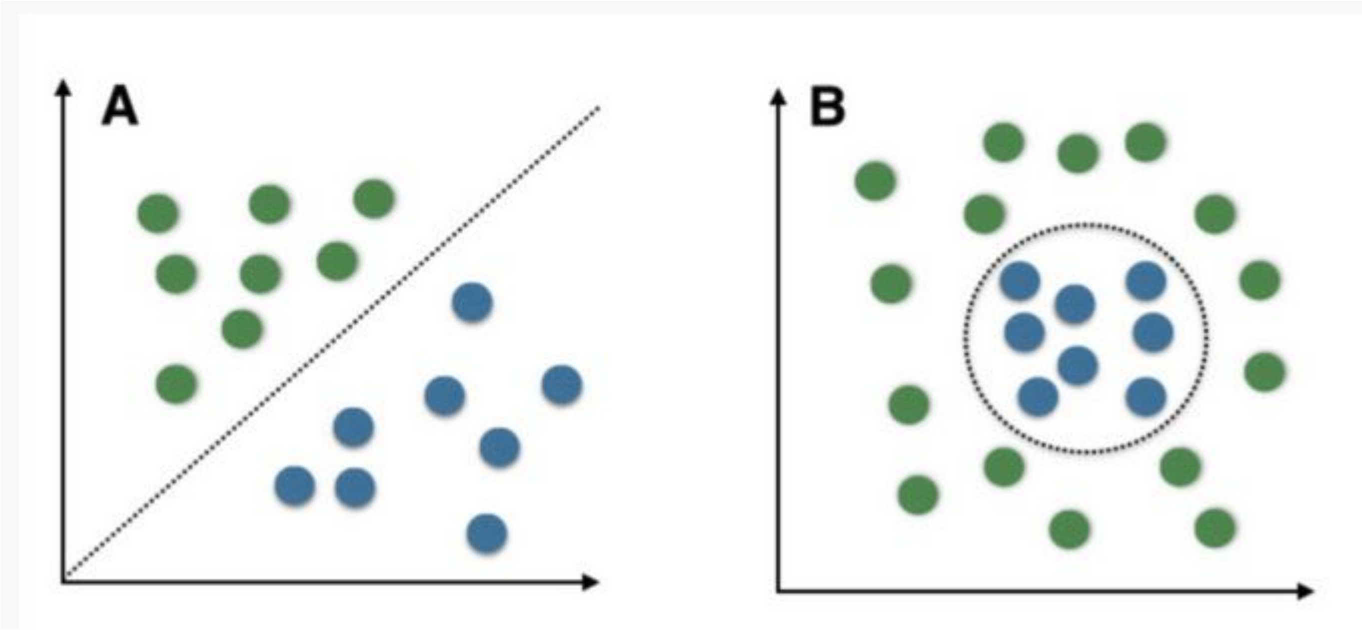


iter=150



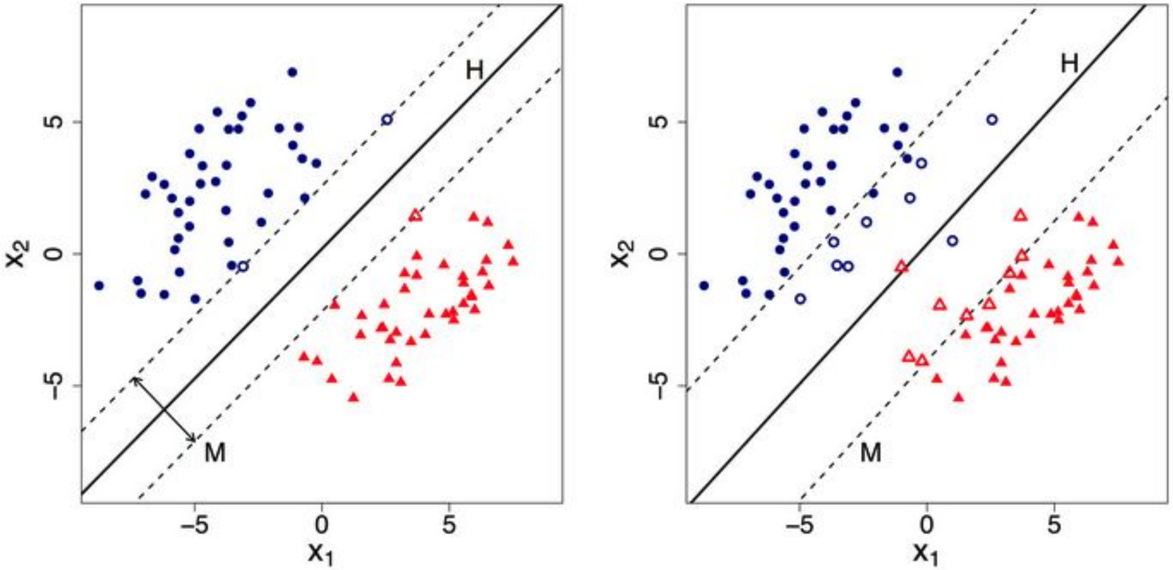
Exercise: implementing logistic regression w gradient descent

Importance of non-linearity



https://www.codingninjas.com/codestudio/library/linear-vs-non-linear-classification

Support vector machines



Hard margin (left) and soft margin (right) SVMs.

https://www.surveypractice.org/article/2715-using-support-vector-machines-for-survey-research

Tree based methods

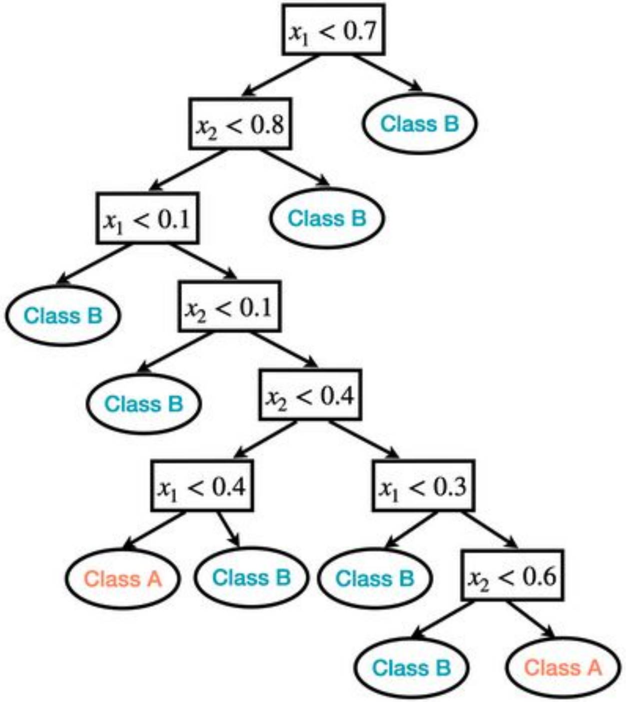


Figure 13.1: An example of classification tree based one two predictors.

https://bookdown.org/hailiangdu80/Machine_Learning_and_Neural_Networks/tree-based-models.html

Neural network: backpropagation

- Gradient descent is used to optimize the parameters for neural networks as well.
- More specifically, **stochastic gradient descent,** is used:
  - This just means gradient descent is done in a ‘random’ portion of the data called the **batch.**
- Backpropagation is used to propagate the gradient back to the layers and update weights.

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \qquad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \qquad \text{(BP2)}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \qquad \text{(BP3)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \qquad \text{(BP4)}$$

http://neuralnetworksanddeeplearning.com/chap2.html

Neural network: backpropagation

Feedforward part:

for i in range(len(layers)):

$$a(i) = o(i-1).w[i]$$

$$o(i) = \frac{1}{1+e^{-a(i)}}$$

$$= sigmoid(a(i))$$

Backpropagate part:

for the output layer :

$$error[L] = o[L] - y$$

$$grad[L] = error * \text{sigmoid\_deriv}(o[L])$$

for other layers :

$$error[i] = grad[i+1].W[i]$$

$$grad[i] = error[i] * \text{sigmoid\_deriv}(o[i])$$

Exercise: looking at different components of the backpropagation algorithm.