Basic ML models

@asmalx

January 2021

1 Regression models

When the target variable that we're trying to predict is continuous, we call the learning problem a regression problem. Generally, regression models investigate relationship between dependent (target) value Y and independent variable (predictor) X. This is supervised learning model, we use training set to fit hypothesis closer to real values.

1.1 Definitions

1. **Data**

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & ... & x_n^{(2)} \\ x_1^{(2)} & x_2^{(2)} & ... & x_n^{(1)} \\ ... & ... & ... & ... \\ x_1^{(m)} & x_2^{(m)} & ... & x_n^{(m)} \end{bmatrix}$$

X is a $m \times n$ matrix, where m - number of training examples, n - number of variables. Represents input data.

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ ... \\ y^{(m)} \end{bmatrix}$$

Y is a $m \times 1$ matrix or m-dimensional vector of data, which we want to predict.

2. **Coefficients**

$$\Theta = \begin{bmatrix} \Theta^{(0)} \\ \Theta^{(1)} \\ ... \\ \Theta^{(n)} \end{bmatrix}$$

Θ is a (n+1)-dimensional vector of coefficients at $f_1(x_1), f_2(x_2), ..., f_n(x_n)$ and the bias coefficient Θ_0. Θ is the value which determined in the learning process, but $f_1, f_2, ..., f_n$ are known.

3. **Hypothesis**

$$h_\Theta(x) = \Theta_0 + \Theta_1 f_1(x_1) + \Theta_2 f_2(x_2) + ... + \Theta_n f_n(x_n)$$

Hypothesis is a function which describes the target, maps $X \to Y$

1.2 Cost function

The cost function is usually used to measure the accuracy of hypothesis. It takes an average difference of $h_\Theta(X)$ from Y.

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

Using matrices:

$$J(\Theta) = \frac{1}{2m}(X\Theta - Y)^T(X\Theta - Y)$$

Also called "Squared error function", "Mean squared error". Obviously, minimizing the cost function, we get the most accurate prediction model.

1.2.1 Implementation in Python using numpy

```
def Cost(theta):
    x = np.ones((X.shape[0],X.shape[1]+1))
    x[:,1:] = X      # add one's column
    h = np.matmul(x,theta) - Y)      #calculate hypothesis
    return np.matmul(np.transpose(h), h)[0,0]  / (2*x.shape[0])
```

1.3 Gradient descent

Gradient descent is a iterative algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent.

repeat until convergence:

$$\Theta_i := \Theta_i - \alpha\frac{\partial}{\partial\Theta_i}J(\Theta)$$

Important! At each iteration parameters $\Theta_1, \Theta_2, ..., \Theta_n$ should be updated simultaneously (using vectorized calculations, for example)

1.4 Linear regression

$$f_i(x_i) = x_i \text{ - due to linearity;}$$

$$h_\Theta(X) = \Theta_0 + \Theta_1 x_1 + ... + \Theta_n x_n$$

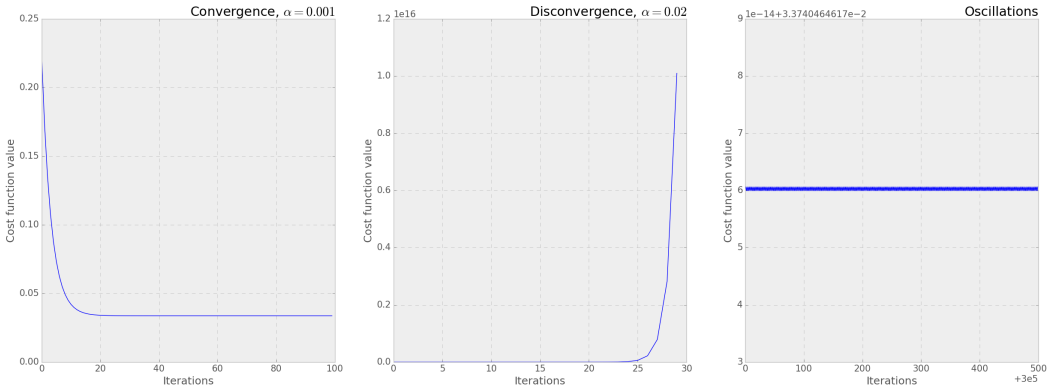
.

1.4.1 Example

Dataset consist of two columns: **vehicle CO2 emission, g/km** and **fuel consumption, L/100km**. We use linear regression model to predict CO2 emission with fuel consumption. Training set formed with 20 samples, dataset size is 7000 samples. First 5 data rows:

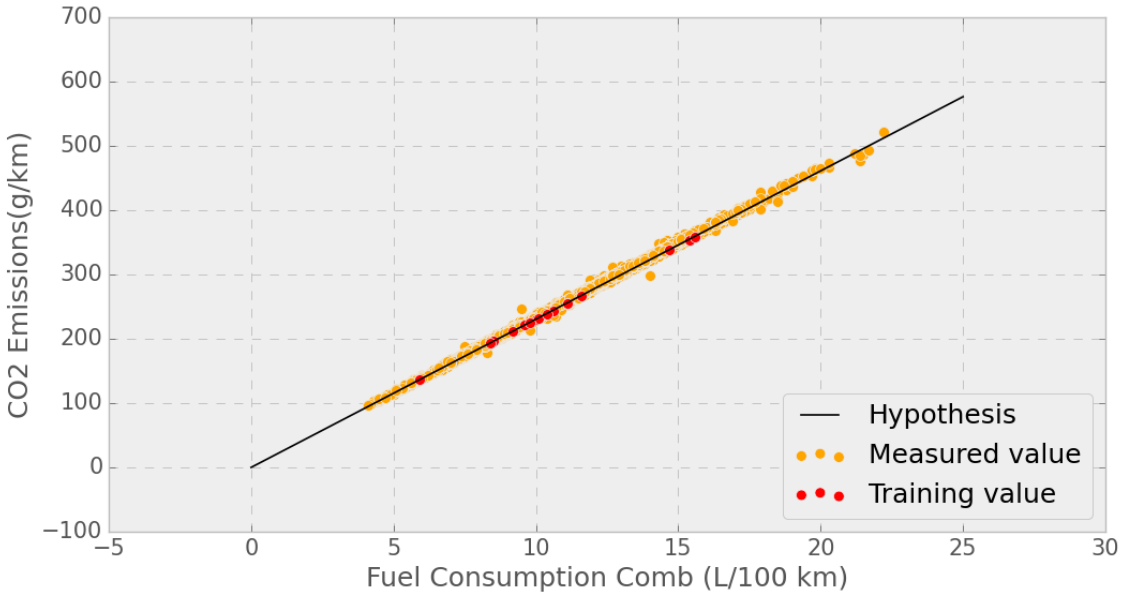
Fuel Consumption Comb (L/100 km)	CO2 Emissions(g/km)
8.5	196
9.6	221
5.9	136
11.1	255
10.6	244

Figure 1: Gradient descent convergence



In gradient descent algorithm parameter α controls descent speed, too small value will require more iterations, but huge α leads to disconvergence: algorithm literally 'jump' throw extremum point. Also it can oscillate around extremum point if α doesn't fit well (on Figure 1). Linear regression model has very good prediction results in

Figure 2: Plotting data



this case: Learned hypothesis shown at Figure 2.

1.5 Polynomial regression

Assume h_Θ as a polynomial function of X. Generally, have $\eta = C^k_{n+k}$ terms in function (k - polynomial degree), therefore calculation for higher degrees can be long/expensive. Example for degree 2 and 2 variables:

$$h_\Theta(X) = \Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_1 x_2 + \Theta_4 x_1^2 + \Theta_5 x_2^2.$$

Redefining $x_{r_1}^{m_1} x_{r_2}^{m_2} ... x_{r_l}^{m_l}$ as t_k we reduce problem to linear regression (got $X \to T$, $\Theta \to \Theta^t$, but if X has dimension $m \times n$, T has dimension $m \times \eta$)

$$h_\Theta(T) = \Theta_0^t + \Theta_0^t t_1 + ... + \Theta_0^t t_\eta$$

1.6 Logistic regression

Logistic regression models used to prediction if y can take on only a binary value; it is a model for classification problem.

1.6.1 New hypothesis

We need to change form of $h_\theta(X)$ to satisfy $0 \leq h_\theta(X) \leq 1$. This is accomplished by plugging $X\Theta$ into the Logistic Function g(z).

$$g(z) = g(X\Theta) = \frac{1}{1+e^{-z}}$$

So, now $h_\theta(X)$ will give the probability that our output is 1.

1.6.2 New cost function

We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local extremums. Instead, let cost function for logistic regression looks like:

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}),$$

$$\text{where } Cost(h_\theta(x), y) = \begin{cases} -\log h_\theta(x) & \text{if y = 1} \\ -\log 1 - h_\theta(x) & \text{if y = 0} \end{cases}$$

If correct answer 'y' is 0, then the cost function will be 0 if hypothesis function also outputs 0. If hypothesis approaches 1, then the cost function will approach infinity and vice versa. Note that writing the cost function in this way guarantees that J(Θ) is *convex* [1] for logistic regression.
Due to fact that $y^{(i)}$ is binary ($y \in \{0,1\}$), $J(\Theta)$ can be written:

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

And vectorized implementation:

$$J(\Theta) = -\frac{1}{m}\left[Y^T \log g(X\theta) + (1-Y)^T \log\left(1-g(X\theta)\right)\right]$$

1.6.3 Gradient descent

Repeat until convergence:

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

And vectorized realization:
Repeat until convergence:

$$\Theta := \Theta - \frac{\alpha}{m}X^T\left[g(X\Theta) - Y\right]$$

1.6.4 Expand to multi-class classification: One-vs-all method

Instead of y = {0,1} let y = {0,1...n}.
Since y = {0,1...n}, we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.
We are basically choosing one class and then merging all the others into a single second class. We do this repeatedly, applying binary logistic regression to each case, and then use the hypothesis that returned the highest value as our prediction.

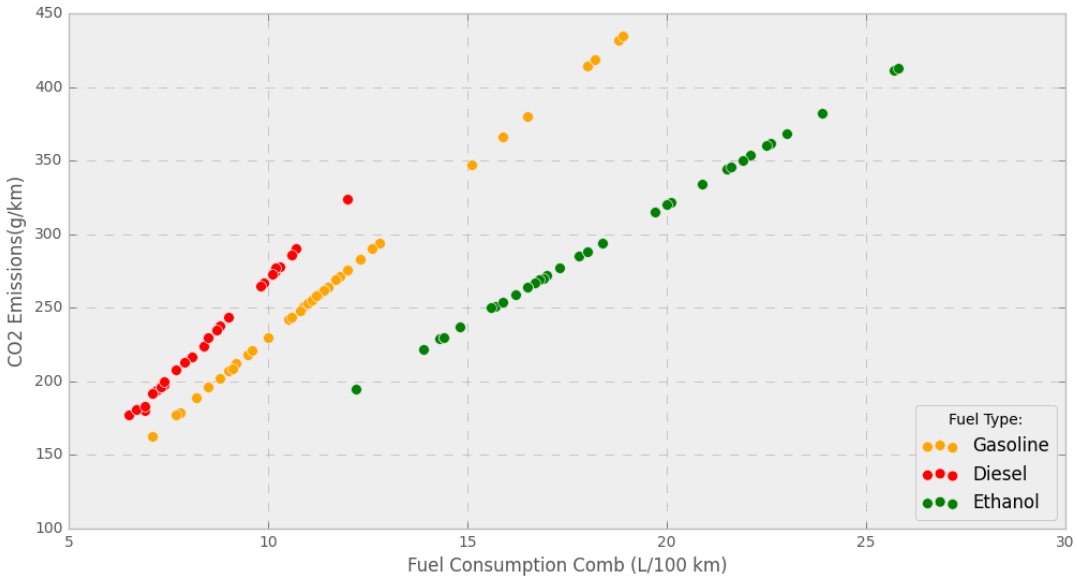
[1] Function is called convex if the line segment between any two points on the graph of the function lies above the graph between the two points. Strictly convex function on an open set has no more than one minimum.

1.6.5 Example

For example of classification problem solving we use the same dataset, but will predict fuel type with fuel consumption and CO2 emission. We will use 50 samples for each fuel type to learn model; chosen polynomial representation of hypothesis with degree = 2. Example of 5 random rows:

Fuel Consumption Comb (L/100 km)	CO2 Emissions(g/km)	Fuel Type
8.5	199	Gasoline
8.4	224	Diesel
10.9	257	Gasoline
18.0	296	Ethanol
9.0	212	Gasoline

Figure 3: Plotting training data set



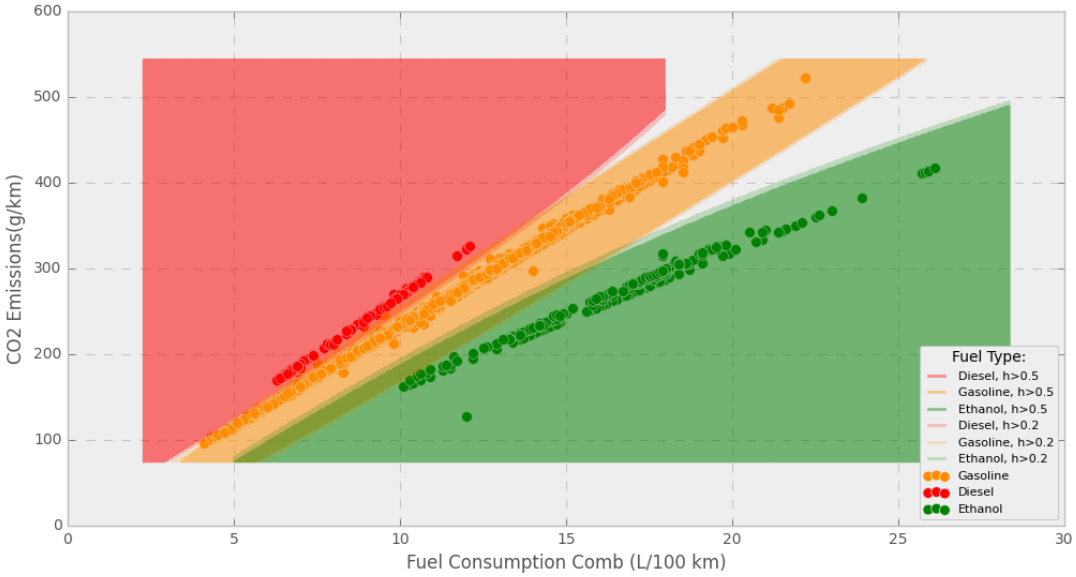
Using "One vs all" principle, create 3 logistic regression models for each class element: for 'gasoline', for 'diesel' and for 'ethanol'. Specific transformation of input data lead hypothesis to linear function ($h_\Theta(X) \to h^t_\Theta(T)$):

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_1x_2^{(1)} & x_1^{2(1)} & x_2^{2(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_1x_2^{(2)} & x_1^{2(2)} & x_2^{2(2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & x_1^{(m)} & x_2^{(m)} & x_1x_2^{(m)} & x_1^{2(m)} & x_2^{2(m)} \end{bmatrix} \to \begin{bmatrix} 1 & t_2^{(1)} & t_3^{(1)} & t_4^{(1)} & t_5^{(1)} & t_6^{(1)} \\ 1 & t_2^{(2)} & t_3^{(2)} & t_4^{(2)} & t_5^{(2)} & t_6^{(2)} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & t_2^{(m)} & t_3^{(m)} & t_4^{(m)} & t_5^{(m)} & t_6^{(m)} \end{bmatrix} = T$$

Before processing, data were normalazed with 'Z-score' [2] nomalization.
Learned model represented on figure 4:

[2]Normalazing - adjusting values with different scales to a some common scale. 'Z-score' nomalization: $x_{norm} = \frac{x-\mu}{\sigma}$, where μ is data average value, σ - standart deviation.

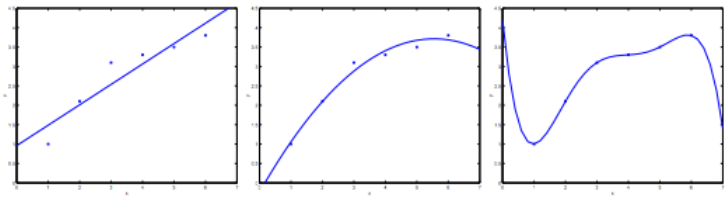
Figure 4: Model after learning



There are three colored polygons, which represent field of hypothesis values: ≥ 0.2 (light color) and ≥ 0.5. Hypothesis values define probability of belonging some engine to a fuel class.

1.7 The Problem of Overfitting

Figure 5: From left to right: underfitting, fitting, overfitting



Underfitting, or high bias, is when the form of our hypothesis function h maps poorly to the trend of the data. It is usually caused by a function that is too simple or uses too few features. At the other extreme, overfitting, or high variance, is caused by a hypothesis function that fits the available data but does not generalize well to predict new data. It is usually caused by a complicated function that creates a lot of unnecessary curves and angles unrelated to the data.
There are two main options to manage the issue of overfitting:

1. Reduce the number of features. Manually select which features to keep. Use a model selection algorithm *.
2. Regularization. Regularization works well when we have a lot of slightly useful features. Keep all the features, but reduce the magnitude of parameters Θ_i by introducing the parameter λ.

$$\text{So, we add this term to the cost function: } + \lambda \sum_{j=1}^{n} \Theta_j^2$$

2 Neural networks

2.1 Neuron model

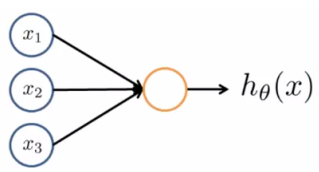


Figure 6: Neuron model

Neurons are computational units that take inputs $x_1, x_2, ..., x_n$ and channel them to output. Often there is 'bias' input added, $x_0 = 1$. In neural networks we use the same logistic function as in classification for neuron, but we call it activation function, and Θ parameter called 'weights'.

Neurons are connected to layers. Input nodes also known as the 'input layer'; layer, which finally outputs the hypothesis function, known as the 'output layer'.All intermediate layers of nodes between the input and output layers called the 'hidden layers'.

$a_i^{(j)}$ - activation of unit i in layer j, $\Theta^{(j)}$ - matrix of weights controlling function mapping from layer j to layer j+1.

2.2 Feedforward neural network

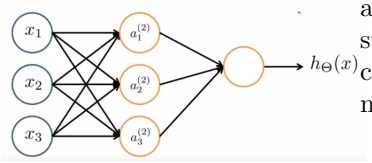


Figure 7: Neural network

NN? There are no feedback neuron's connections in which outputs of the model are fed back into itself. In feedforward neural networks input data are processed step by step through layers to the output value. With neural wetwork we can get complicated, nonlinear relationship between y and x, using only set of connected neurons as logistic regression models.

Let k - number of inputs for layer j, m - number of neurons in layer j. Then:

$$a_i^{(j)} = g(\Theta_{i,0}^{(j-1)} a_0 + \Theta_{i,1}^{(j-1)} a_1^{(j-1)} + \Theta_{i,2}^{(j-1)} a_2^{(j-1)} + ... + \Theta_{i,k}^{(j-1)} a_k^{(j-1)})$$

Vectorized implementation:

$$a^{(j)} = g\left(\begin{bmatrix} \Theta_{1,0}^{(j-1)} & \Theta_{1,1}^{(j-1)} & ... & \Theta_{1,k}^{(j-1)} \\ \Theta_{2,0}^{(j-1)} & \Theta_{2,1}^{(j-1)} & ... & \Theta_{2,k}^{(j-1)} \\ ... & ... & ... & ... \\ \Theta_{m,0}^{(j-1)} & \Theta_{m,1}^{(j-1)} & ... & \Theta_{m,k}^{(j-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1^{(j-1)} \\ ... \\ a_k^{(j-1)} \end{bmatrix}\right) = g\left(\Theta^{(j-1)} a^{(j-1)}\right)$$

2.2.1 Cost function

For neural network cost function looks:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\theta(x^{(i)})_k) + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\left(\Theta_j^l\right)^2$$

where K - count of object to classify, L - count of neurons in last layer, s_l - count of neurons in l layer. (Generally, K=L)

2.2.2 Backpropagation algorithm

'Backpropagation' is neural network terminology; it is technique for minimizing cost function with efficient way, just like gradient descent for regression problems. General concept!!!

$$J(\Theta) = J((a_L, y),\ a_L = g(z_L),\ z_L = \Theta^{(L)} a_{L-1}, \Rightarrow \frac{\partial J(\Theta)}{\partial \Theta_j} = \frac{1}{m}\sum_{i=1}^{s_L} \frac{\partial J(\Theta)}{\partial a_L}\frac{\partial a_L}{\partial z_L}\frac{\partial z_L}{\partial \Theta_j}$$

1. There is the derivative of cost function
2. $$\frac{\partial a_L}{\partial z_L} = \frac{\partial g(z_L)}{\partial z_L} = g'(z_L) = \frac{-e^{-z_L}}{(1 - e^{-z_L})^2} = \frac{1}{(1 - e^{-z_L})} - \frac{1}{(1 - e^{-z_L})^2} = g(z_L) \cdot (1 - g(z_L))$$

3.

$$\frac{\partial z_L}{\partial \Theta_j} = \frac{\partial(\Theta_{L-1} \cdot a_{L-1})}{\partial \Theta_j} = a_{L-1}$$

So, we get:

$$\frac{\partial J(\Theta)}{\partial \Theta_j} =$$

2.2.3 Gradient approximation

Often it needs to check value of gradient to be sure backpropagation works as intended. Derivative can be approximated in the following way:

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \Theta_2, ..., \Theta_j + \varepsilon, ..., \Theta_n) - J(\Theta_1, \Theta_2, ..., \Theta_j - \varepsilon, ..., \Theta_n)}{2\varepsilon}$$

For properly working, ε should be small enough, recommended value: $\varepsilon \approx 10^{-4}$