Mahdi Ranjbar
April 16, 2022

Algorithmic Project Report

Implementation of different machine learning models

Having developed a keen eye for the emerging technologies around me, I have always been curious to discover how we are able to help different industries such as financial services, health care, manufacturing and so on, using AI application. The main reasons I decided to work on this algorithmic project were at first, to challenge myself to learn different algorithms and develop my mathematical skills and at second, to obtain an adequate vision of AI world.

Linear Regression

My project was started with *Linear Regression* to predict profits for a business based on populations from cities. Using a dataset with only one variable let visualize different steps along implementation of the model. The objective of *Linear Regression* is to minimize the cost function :

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

where the hypothesis is given by the linear model :

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

One way to minimize cost by adjusting parameters of the model (the theta values) is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update :

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \qquad \text{(simultaneously update } \theta_j \text{ for all } j)$$

The computed cost function with founded theta by gradient descent algorithm was *4.48*.

Then, *Regularized Linear Regression* was implemented on the same dataset. The objective of *Regularized Linear Regression* is to minimize the cost function :

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

It can be noticed that the parameter $theta_0$ is not regularized.

The hypothesis is given by the linear model :

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1$$

The gradient of the cost function is a vector where the j^{th} element is defined as follows :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)} \qquad \text{for } j = 0,$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1,$$

The computed cost function with founded theta by gradient descent algorithm using the regularization rate equals to *1000* was *6.25*. It can be noticed that this model is less prone to overfitting.

Then, *Linear Regression with multiple variable* was implemented to predict the prices of houses considering size of the house and number of bedrooms. It was noted that house sizes were about *1000* times the number of bedrooms by looking at the values. Therefore, *Feature Scaling* was performed -the mean value of each feature was zero and the standard deviation was one- in order to make gradient descent converge much more quickly. After finding the best learning rate by testing different values and drawing theirs curves, the model was trained using gradient descent algorithm.

Thereafter, *Normal Equation* was implemented on the same dataset of house price prediction. *Normal Equation* is an analytical approach used for optimization which is an alternative for Gradient descent. *Normal Equation* performs minimization without iteration. *Normal Equation* are equations obtained by setting equal to zero the partial derivatives of the sum of squared errors or cost function. It allows one to estimate the parameters of multiple linear regression using this formula :

$$\theta = (X^T X)^{-1} X^T \overrightarrow{y}$$

Given a test example, it was noticed that the predicted value using gradient descent algorithm was exactly the same as the one using *Normal Equation*.

Logistic Regression

Afterwards, L*ogistic Regression* model was implemented to predict whether a student gets admitted into a university based on its results on two exams. The cost function in logistic regression is :

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

where hypothesis is defined as :

$$h_\theta(x) = g(\theta^T x)$$

And g is the sigmoid function. The sigmoid function is defined as :

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

And the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0, 1, … , n) defined as follows :

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

After fitting the parameters using gradient descent algorithms, computed accuracy on training set was 89%.

Afterwards, *Regularized Logistic Regression* was implemented to predict whether cylinder engines from a manufacturing plant pass quality assurance according to their results on two different test. By visualizing the dataset, it was seen that the dataset cannot be separated into positive and negative examples by a straight-line through the plot. Therefore, the features were mapped into all polynomial terms of x_1 and x_2 up to the sixth power. As a result of this mapping, the vector of two features (the scores on two quality assurance tests) was transformed into a 28-dimensional vector. The regularized cost function in logistic regression is :

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

It can be noticed that $theta_0$ is not regularized.

The gradient of the cost function is a vector where the j^{th} element is defined as follows :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \qquad \text{for } j = 0,$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \qquad \text{for } j \geq 1,$$

After fitting the parameters by gradient descent algorithms using learning rate equals to one, computed accuracy on training set was 83.05%.

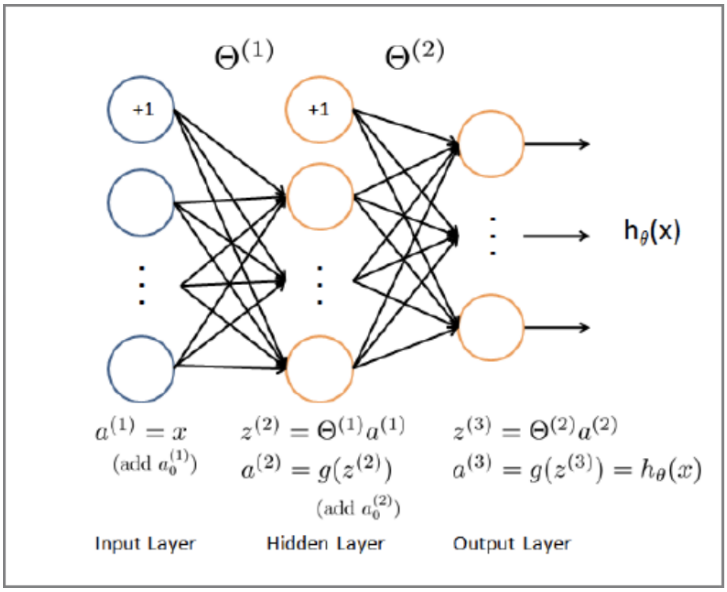
Multi-class Classification

Thereafter, *One-As-All Logistic Regression* model was implemented to build a multi-class classifier. To do so, a subset of *MNIST* dataset was taken in order to predict the digit contained in a given image. Each training example was a 20 pixel by 20 pixel grayscale image of the digit. Each pixel was represented by a floating point number indicating the grayscale intensity at that location. Then, dataset was split into 80% training set and 20% test set.

Since there were 10 classes, it was needed to train 10 separate logistic regression classifiers. After training the *One-Vs-All Classifier,* the computed accuracy on training set was 95.02% and the one on test set was 91.05%.

Neural Network

Subsequently, a 3 layer neural network was implemented to recognize handwritten digits using the same training set as before.



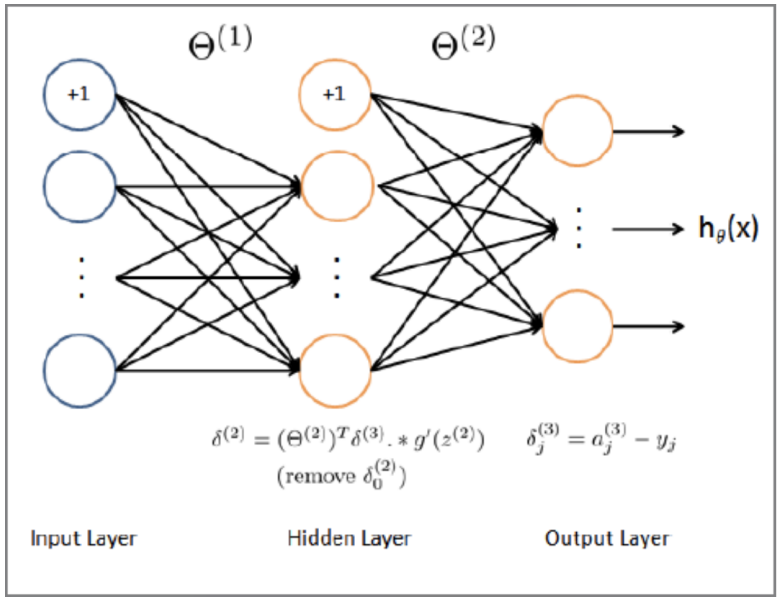
The cost function for neural networks with regularization is :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)})_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}\left(\Theta_{j,k}^{(1)}\right)^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}\left(\Theta_{j,k}^{(2)}\right)^2\right]$$

Then in order to compute the gradient of the loss function with respect to weights of the network for a single input–output, *Back-Propagation Algorithm* was implemented.

The intuition behind the *Back-Propagation Algorithm* is as follows. Given a training example ($x^{(t)}$, $y^{(t)}$), first a *'forward pass'* is run to compute all the activations throughout the network, including the output value of the hypothesis $h_{theta}(x)$. Then, for each node *j* in layer *l*, an 'error term' $delta^{(l)}_j$ is computed that measures how much that node was 'responsible' for any errors in our output.

For an output node, it can be directly measured the difference between the network's activation and the true target value, and it's used to define $delta^{(3)}_j$ (since layer 3 is the output layer). For the hidden units, $delta^{(l)}_j$ is computed based on a weighted average of the error terms of the nodes in layer *(l+1)*.



In detail, the *Back-Propagation Algorithm* can be implemented as follows :

1. Set the input layer's values $(a^{(1)})$ to the t^{th} training example $x^{(t)}$. Perform a *feedforward pass*, computing the activations $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$ for layers 2 and 3.

2. For each output unit k in layer 3 (the output layer), set $delta^{(3)}_k = (a^{(3)}_k - y_k)$ where y_k indicates whether the current training example belongs to class k $(y_k = 1)$, or if it belongs to a different class $(y_k = 0)$.

3. For the hidden layer $l=2$, set $$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

4. Accumulate the gradient from this example using the following formula :

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

5. Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $1/m$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

6. Add regularization to the gradient :

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \text{for } j = 0,$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \text{for } j \geq 1$$

After implementation of *Back-Propagation Algorithm,* the gradient descent with regularization rate equals to one was used to fit the parameters. The computed accuracy on training set was 95.45% and the one on test set was 92.60%.

Regularized Logistic Regression Application

Finally, a data set which contains information related to red wine was taken in order to predict whether the wine is of good or bad quality using *Regularized Logistic Regression* already implemented.

At the start, the label coding was done as follows, '*bad*' quality was encoded as zero and *'good'* quality as one. Then, a *Cross-Validation* was performed in order to split the dataset into 3 parts as follows : 60% training set, 20% validation set, 20% test set.

Thereafter, by looking at the values of features, it was noted that there are many differences between orders of magnitudes. Therefore, *Feature Scaling* was performed. Subsequently, in order to find the best learning rate, some models with different learning rate were trained in parallel. Then only the one was selected that worked better at the end on the validation set. Same as finding the best learning rate, in order to find the best regularization rate, some models with different regularization rate were trained in parallel and the one that worked the best on our validation set was selected finally.

At the end, the computed accuracy using the best learning rate and regularization rate on training set was 74.06% and the one on validation set was 78.43% and lastly the one on test set was 74.29. It was concluded that the model suffers from under-fitting, in other words the model is high-bias and low-variance. This problem would have been solved by adding polynomial features or using a more complex model such as Decision Tree.