Chapter 5: Supervised Learning using PyTorch

**Introduction**

Supervised machine learning is the most sophisticated branch of machine learning that is in use in almost all the fields including artificial intelligence, cognitive computing and language processing. Broadly the machine learning literature talks about three types of learning supervised, unsupervised and reinforcement learning. In supervised the machine learns to recognize the output, hence it is task driven and the task can be classification or regression.

In unsupervised learning the machine learns patterns from data and hence generalizes to the new dataset and the learning happens by taking a set of input features. In reinforcement learning the learning happens in response to a system that react too situations. In this chapter we are going to cover regression techniques in detail with the machine learning approach and interpretation of output from each of the regression methods in the context of a business scenario.

In real world scenario there are cases where a regression is not going to help us in predicting the target variable. In supervised regression technique the input data is also known as training data, for each record there is a label which is a continuous numerical value. The model is prepared through training process in predicting the right output and the process continues till the desired level of accuracy is achieved. We may need advanced regression methods to understand the pattern existing in the data set.

# Introduction to Linear Regression

Linear regression analysis is known as the most reliable and easy to apply that is widely used of all statistical techniques. This assumes linear, additive relationships between dependent and independent variables. The objective of linear regression is to predict the dependent or target variable through independent variables. The specification of the linear regression model is as below.

Y = α + βX

This formula has the property that the prediction for Y is a straight-line function of each of the X variables, keeping all others fixed, and the contributions of different X variables to the predictions are additive. The slopes of their individual straight-line relationships with Y are the coefficients of the variables. The coefficients and intercept are estimated by least squares, i.e., setting them equal to the unique values that minimize the sum of squared errors within the sample of data to which the model is fitted. The model's prediction errors are typically assumed to be independently and identically normally distributed. When the beta coefficient becomes zero the input variable X has no impact on the dependent variable. The OLS methods attempts at minimizing the sum of squared residuals, the residuals are defined as the difference between the points on the regression line to the actual data points in the scatterplot. This process seeks to estimate the beta coefficients in a multiple linear regression model.

There is always a question comes to our mind that why do we assume that there exists a linear relationship between the dependent variable and a set of independent variables, when most of the real-life scenarios reflect any other relationship but linear relationship. The reasons why we still stick to linear relationship are as below:

This is easy to understand and interpret, there are ways to transform the existing deviation from linearity and make it linear, Simple to generate prediction.

In Machine learning to be more precise in the field of predictive modeling is mainly concerned with minimizing the error of a predictive model or making the most accurate predictions possible. Linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables but has been borrowed by machine learning. It is both a statistical algorithm and a machine learning algorithm. The linear regression model depends on a set of assumptions such as:

Linear relationship between dependent and independent variables

No multicollinearity among the predictors

There should not be any autocorrelation

There should not be any heteroscedasticity

Within linear regression there are different variants but in machine learning we consider them as one method. For example, if we are using one explanatory variable to predict the dependent variable it is called simple linear regression model. If we are using more than one explanatory variable, then the model is called multiple linear regression models. The ordinary least square is a statistical technique used to predict the linear regression model; hence sometimes the linear regression model is also known as ordinary least square model.

Linear regression is very sensitive to missing values and outliers the reason being the statistical method of computing a linear regression depends on mean, standard deviation and covariance between the variables. Mean is sensitive to outlier values; hence it is expected that we need to clear out the outliers before proceeding towards forming the linear regression model. In machine learning literature the method of getting optimum beta coefficients that minimizes the error in a regression model can be achieved by a method called gradient descent algorithm. How the gradient descent algorithm works? It starts with an initial value preferably from zero and updates the scaling factor by a learning rate regularly iteratively to minimize the error term.

Linear regression understanding based on a machine learning approach requires special data preparation that avoids some assumptions directly by keeping the original data intact. Data transformation is required in order to make your model more robust.

## Problem:

How to perform data preparation for creating a supervised learning model using PyTorch?

## Solution:

We will take an open source data set mtcars.csv, which is a regression dataset, to test out how to create input and output tensor.

## How It works:

First, the necessary library needs to be imported.

A screenshot of a cell phone

Description generated with very high confidence

The predictor for the supervised algorithm is qsec and it is used to predict the mileage per gallon provided by the car. What is important here is data type. First, we import the data which is in numpy format, into a PyTorch tensor format. Default tensor format is float, using the tensor float format would give error while performing the optimization function. Hence it is important to change the tensor data type. Using unsqueeze function and specifying the dimension equal to 1, we can reformat the tensor type.

A screenshot of a cell phone

Description generated with very high confidence

To reproduce the same result a manual seed need to be set. Hence the torch.manual\_seed(1234) was used. Though we see that data type is tensor, if we check the type function it will show as Double, as tensor type double would be required for the optimization function.

A screenshot of a cell phone

Description generated with very high confidence

## Problem:

How do you use build a neural network torch class function, so that you can build forward propagation method?

## Solution:

Design the neural network class function including the hidden layer from input layer and from hidden to output layer. In the neural network architecture number of neurons in the hidden layer also need to be specified.

## How It works:

In the class Net() function we initialize first the feature, hidden and output layer. Then we will introduce the back-propagation function using the rectified linear unit as the activation function in the hidden layer.

A screenshot of a cell phone

Description generated with very high confidence

The following image shows the ReLU activation function.

A screenshot of a cell phone

Description generated with very high confidence

Now the network architecture is mentioned as below for the supervised learning model. The n\_feature shows the number of neurons in the input layer. As we have one input variable qsec, we will use 1, the number of neurons in the hidden layer can be decided based on the input and degree of accuracy required in the learning model. We use the n\_hidden equal to 20, which means 20 neurons in the hidden layer 1. And the output neuron would be 1.

A screenshot of a cell phone

Description generated with very high confidence

The role of the optimization function is to minimize the loss function defined with respect to the parameters and the learning rate. The learning rate chosen here is 0.2, and we also pass the neural network parameters into the optimizer. There are various optimization functions.

* SGD: this function Implements stochastic gradient descent (optionally with momentum), the parameters could be momentum, learning rate and weight decay.
* Adadelta: adaptive learning rate. It has five different arguments, parameters of the network, coefficient used for computing a running average of the squared gradients, addition of a term for achieving numerical stability of the model, learning rate and weight decay parameter to apply regularization
* Adagrad: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, this has arguments like iterable of parameter to optimize, learning rate and learning rate decay with weight decay
* Adam: it is a method for stochastic optimization. This function has six different arguments, iterable of parameters to optimize, learning rate, betas, which is known as coefficients used for computing running averages of gradient and its square, a parameter to improve numerical stability etc.
* ASGD: Acceleration of stochastic approximation by averaging, this has five different arguments, iterable of parameters to optimize, learing rate, decay term, weight decay etc.
* RMSProp algorithm
* SparseAdam: Implements lazy version of Adam algorithm suitable for sparse tensors. In this variant, only moments that show up in the gradient get updated, and only those portions of the gradient get applied to the parameters.

Apart from the optimization function, there is a loss function that need to be selected before running the supervised learning model. Again, there are various loss functions, let’s look at the error functions.

* MSELoss: Creates a criterion that measures the mean squared error between elements in the input variable and target variable.
* For regression related problem this is the best less function that get’ s utilized

A screenshot of a cell phone

Description generated with very high confidence

After running the supervised learning model, which is a regression model we need to print the actual vs. predicted values and represent them in a graphical format. Hence, we need to turn on the interactive feature of the model.

## Problem:

How to build a basic supervised neural network training model using PyTorch with different iterations.

## Solution:

Basic Neural Network Model in PyTorch requires six different steps such as preparation of training data, initialization of weights, creating a basic network model, calculating loss function, selecting the learning rate and finally optimizing the loss function with respect to the parameters of the model.

## How It works:

Let’s follow a step by step approach to create a basic neural network model as stated above.

A screenshot of a cell phone

Description generated with very high confidence

Result:

The final prediction result from the model with first iteration and the last iteration is now represented in a graph as below.

A screenshot of a social media post

Description generated with very high confidence

In the initial step the loss function was 276.91 and after optimization the loss function becomes 35.1890. the fitted regression line and how it is fitted to the data set is represented above.

## Problem:

How to extract best results from the PyTorch based supervised learning model.

## Solution:

The computational graph network is represented by nodes and connected through functions. There are various techniques that can be applied to minimize the error function and get the best predictive model. We can increase the iteration numbers, estimate the loss function, optimize the function, print actual and predicted values and show it in a graph.

## How It works:

To apply tensor differentiation nn.backward() method need to be applied. Let’s take an example and see how the error gradients are back propagated. The. grad () function will hold the final output from the tensor differentiation.

A screenshot of a cell phone

Description generated with very high confidence

The tuning parameters that can increase the accuracy of the supervised learning model, which is a regression use case can be achieved by the following methods.

* Number of iterations
* Type of loss function
* Selection of optimization method
* Selection of loss function
* Learning rate
* Decay in the learning rate
* Momentum require for optimization or not

A screenshot of a cell phone

Description generated with high confidence

Real dataset looks like this as below:

A screenshot of a cell phone

Description generated with very high confidence

A screenshot of a social media post

Description generated with very high confidence

After 1000 iterations the model converges.

A screenshot of a cell phone

Description generated with very high confidence

The neural networks in the torch library typically is being used along with the nn module, let’s took a look at that.

Neural networks can be constructed using the torch.nn package.

Provides pretty much all neural network related functionalities such as :

* Linear layers - nn.Linear, nn.Bilinear
* Convolution Layers - nn.Conv1d, nn.Conv2d, nn.Conv3d, nn.ConvTranspose2d
* Nonlinearities - nn.Sigmoid, nn.Tanh, nn.ReLU, nn.LeakyReLU
* Pooling Layers - nn.MaxPool1d, nn.AveragePool2d
* Recurrent Networks - nn.LSTM, nn.GRU
* Normalization - nn.BatchNorm2d
* Dropout - nn.Dropout, nn.Dropout2d
* Embedding - nn.Embedding
* Loss Functions - nn.MSELoss, nn.CrossEntropyLoss, nn.NLLLoss

Another version of supervised learning algorithm would be standard classification algorithm. Where the target column is a class variable and the features could be numeric as well as categorical.

## Problem:

How to deploy a logistic regression model using PyTorch?

## Solution:

The computational graph network is represented by nodes and connected through functions. There are various techniques that can be applied to minimize the error function and get the best predictive model. We can increase the iteration numbers, estimate the loss function, optimize the function, print actual and predicted values and show it in a graph.

## How It works:

To apply tensor differentiation nn.backward() method need to be applied. Let’s take an example and see how

A screenshot of a cell phone

Description generated with very high confidence

Data preparation for performing logistic regression model.

A screenshot of a cell phone

Description generated with very high confidence

Let’s look at the sample dataset for classification.

A screenshot of a social media post

Description generated with very high confidence

Setting up of the neural network module for logistic regression model.

A screenshot of a cell phone

Description generated with very high confidence

Checking the neural network configuration.

A screenshot of a cell phone

Description generated with very high confidence

Running iterations and finding out best solution for the sample graph.

A screenshot of a cell phone

Description generated with high confidence

The first iteration provides almost 99% accuracy and subsequently the model provides 100% accuracy on the training data.

A close up of a map

Description generated with high confidence

Figure 1: Initial Accuracy

A screenshot of a cell phone

Description generated with high confidence

Figure 2: final accuracy

**Conclusion:**