Task #3

Supervised learning, unsupervised learning, linear regression [cost function and batch gradient descent]

Introduction:

Machine learning is the science of getting computers to learn without being explicitly programmed. Machine learning is now a well-developed domain, with its impact all around in our lives, be it in targeted marketing, natural language processing, recommender systems, computer vision, to name a few. With its application being extended into achieving artificial intelligence and self-driven cars, machine learning is now more relevant than ever. But how do you train the computers to navigate, let alone model, the complex and dense data that they have to process, to achieve accurate predictions? How can a machine learn from and then predict our patterns of our technology lifestyle?

To understand how machine learning works, it is important to understand the data that is processed by these algorithms. All data comes in two forms: either we know that for a dataset, there is a corresponding "right" answer that can be associated with the data. For instance, consider a dataset where we have price of houses corresponding to their area. For each data of "area" we have a corresponding answer of price of house. The training of models on this data is called **supervised learning**. The second form that a dataset can be received in is where we have a compilation of values, and we don't really know how to interpret it, and require to find trends in it. The training of models on this type of data is called **unsupervised learning**.

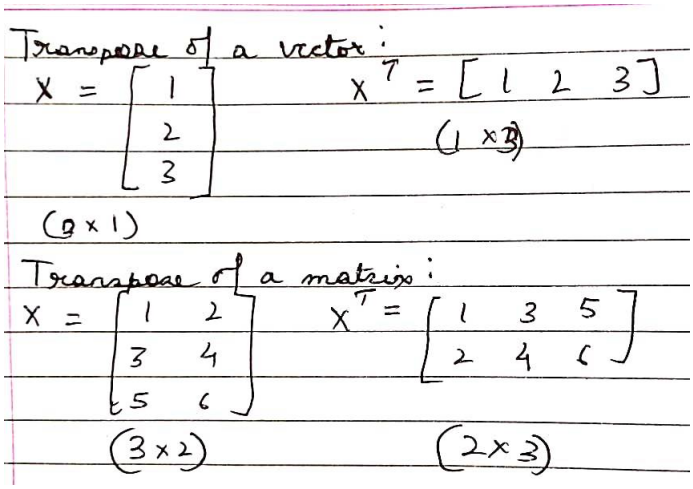
Let's have a deeper understanding of supervised learning and unsupervised learning through the means of examples. Let's consider that we need to predict whether a tumor is benign or malignant based on its size. We would have a dataset which contains several instances of size of tumor versus its diagnosis as benign or malignant. Here, we have an idea of a "right" answer and are looking to generate more of these "right" answers via the trained model. This is an instance of supervised learning algorithms, where we receive a dataset of "right" answers and expect to generate more of them. Now consider a dataset received by the marketing team regarding data of consumer's shopped goods. Here, we just have data of consumers and what they have bought, but we need to understand the patterns in the data to understand which consumers are alike and why. This is an example of unsupervised learning; by the means of the clustering algorithms, it is possible to find alike consumers and their shopping trends for several applications, mainly targeted marketing.

Having understood the two types of learning models, we are now at the position to approach the most basic machine learning algorithm of supervised learning type: **the linear regression model**. But to understand its implementation, the mathematics linear algebra is essential.

Linear algebra:

Linear algebra deals primarily with vectors and matrices, and using their techniques to find solutions to equations. The first concept heavily used in machine learning the transpose of a vector or a matrix. Transpose is the converting all the rows into columns and vice-versa to get inverted dimensions of the same matrix, finding its applications in computing final solution and the concept of vectorization.

Task #3



In machine learning, the first implementation comes in solving linear equations to find solutions. Suppose we have a vector θ [2x1], as the coefficients to the following equation:

$$f(x) = \theta_0 x_0 + \theta_1 x_1$$

And a vector X [2x1], that holds the values whose answer we would like to compute. We simply take the transpose of θ and do multiplication of the vectors to find the final answer:

$$\theta = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad X = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \theta^T = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$f(x) = \theta^T X = 2$$

Note that we need the row dimensions of X and θ to be equivalent for successful computation.

The second implementation is through the technique called vectorization. In many algorithms in machine learning, it is important to find the multiplication of two matrices' element wise values. Instead of iterating through each element, we instead directly multiply the matrices together and make use of efficient linear algebra libraries that compute the output matrices in the blink of an eye, making the code very short, readable and efficient. Note that in most cases, it is important to verify the dimensions of the output matrix, so a matrix transpose is advised, or in the case the multiplication is not possible.

Linear regression [the cost function and gradient descent]:

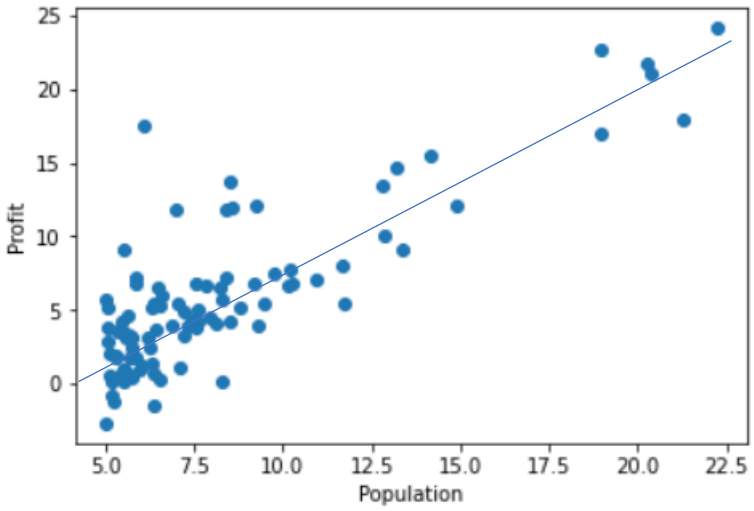
We can now approach the linear regression concept: the simplest implementation of all the machine learning algorithms. To understand linear regression, dissect its term into its components: linear and regression. Linear refers to finding a single line that is the best approximation of the linear [straight-lined] trend in the data, while regression refers to a continuous valued output.

We will define a problem statement, to find the size of profit of a food truck of a restaurant chain, based on the population of a city, through which we can define the linear regression algorithm. Why linear regression is the recommended approach for this problem statement is because we need to generate the possible profit on the dataset; which is never a discrete value.

Task #3

For the entirety of this problem, we will use the dataset provided in the course, “Machine Learning by Stanford on Coursera”, and the code implementation is done on Python. In most machine learning implementations, we divide this data set into two parts: the much larger training set, and the smaller section of test set. We train our algorithm for the training set, and use the test set to verify the results of the algorithm. For this implementation, this will not be done for simplicity in implementation.

How we can approach this problem is demonstrated simply through the graph. We simply find the best-fit line and for any arbitrary ‘x’ or population, we can map its corresponding ‘y’ value or profit to find its value according to the previous trend.



So, how do we find this best fit line with our dataset?

There are some basic terminologies to be defined:

- x(i) = refers to the “input” variable or feature
- y(i) = refers to the output variable or feature
- (x(i), y(i)) = the i^{th} dataset example

h(x) = it refers to the “hypothesis”, the function that takes an input x and tries to output its estimated value of y.

For linear regression, we define it to be as follows: h(x) = θ0 + θ1.x, suggesting that y is a straight-line function of x. Since we have one variable, it is linear regression in one variable.

We now define the cost function: it is the indicator function that let’s us know to fit the best possible straight line to our data. As we know, if we want to fit our best fit line, what we need is to define the θ parameters of the line. To choose its optimum value is to find the most accurate answer possible with this algorithm. So, to choose these values, our idea is to choose them in such a way that the corresponding ‘y’ values of the parameters yield closest to out training set.

Therefore, we have a minimization problem on our hands. We have to minimize θ_0 and θ_1 such that h(x)-y = 0. We take the mean squared approach, where we minimize $(h(x)-y)^2$:

$$\underset{\theta_0 \theta_1}{\text{minimise}} \quad \frac{1}{2m} \times \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2$$

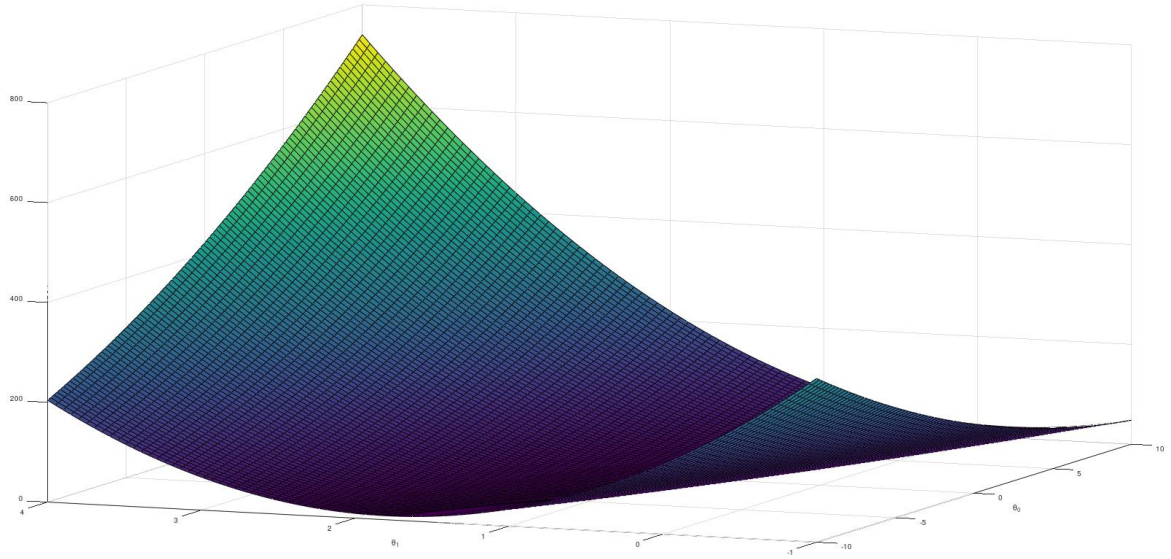
$$\text{where } h(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

Task #3

The code implementation of this function, taking in as inputs X as input feature, y as output feature and theta:

```
def computeCost(X, y, theta):
    J = (1/(2*m))*(np.sum(np.square((X @ theta) - y)))
    return J;
```

The cost function thus represents the error deviations from training examples. To minimize this cost function, which is a function of the θ parameters, we just have to find its global minima. Because we now have two variables, we have a bowl-shaped graph:



An algorithm called gradient descent is used for finding the global minima. Largely just a trial and method error, it is an iterative algorithm:

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

Where alpha is the learning rate, corresponds to how huge a step the algorithm takes in the graph. Bigger alpha, more aggressive gradient descent, however, too big of an alpha result in overshooting the minima and completely missing it. Obvious from the equation, we always have positive value for the alpha.

The code implementing the gradient descent algorithm, taking in as inputs X as input feature, y as output feature, theta, alpha, number of iterations to run the gradient descent and number of training examples. In our code, we will simultaneously update the θ values, which is the correct implementation of the algorithm. While implementing the code, we always initialize the values of θ to zero by convention.

```
def gradientDescent(X, y, theta, alpha, iterations, m):
    J_history = np.zeros(iterations).reshape(iterations, 1)
    for iter in range(iterations):
        theta = theta - ((alpha/m)*(X.transpose() @ ((X @ theta)-y)))
        J_history[iter] = computeCost(X, y, theta)
    return [theta, J_history]
```

Task #3

Say for instance, we reach the global minima. Since the derivative value at the minima is zero, it will leave θ values unchanged, therefore, convergence is guaranteed once found. Note that it can also converge to a local minimum, so several iterations maybe run to find the best solution.

The equation of the derivative of the cost function for both θ values is as follows:

$$\theta_0 \longrightarrow \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 \longrightarrow \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right) \cdot x_1^{(i)}$$

We also call this gradient descent “batch” gradient descent because our computing has the entire training set involved, so it is done over the entire ‘batch’. Mini-batch and stochastic gradient descent also exist, but are out of scope for this paper.

We get this as the output and the corresponding values of θ:

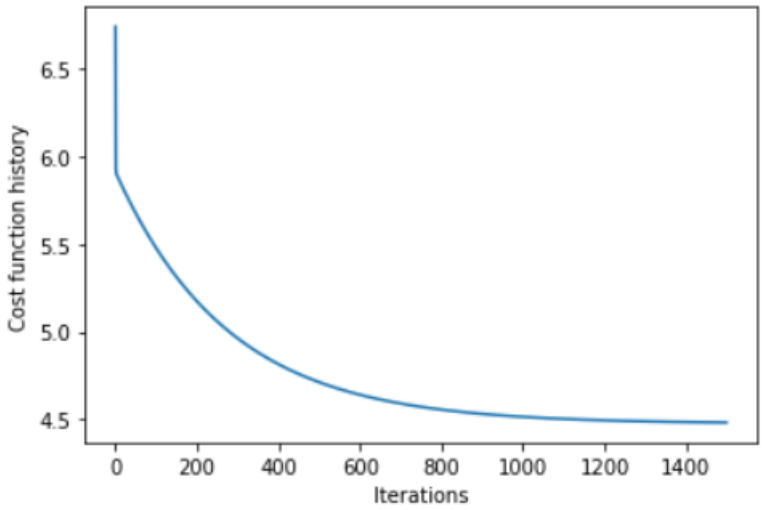
```
The theta values after computing is:
theta0: [-3.63029144] theta1: [1.16636235]
```

Testing these values for arbitrary populations to get the prediction of profit:

```
Prediction of profit for a population of 70,000: [[45342.45012945]]
Prediction of profit for a population of 35,000: [[4519.7678677]]
```

c

We can also see that after each iteration, how much cost function value is being reduced, and as it approaches to minima, its rate of change decreases and stabilizes after a certain number of iterations:



Task #3

Conclusion:

In this paper, we explored machine learning, specifically the different types of learning, supervised and unsupervised, with concrete examples. Then we explored the theory, equations and implementation of machine learning on a real-life problem statement, and studied the cost function and gradient descent through the basics of linear algebra established.

Machine learning is a domain with countless application, and it is the technology that is ruling the earth. We find it everywhere, spam detection, fraudulent activity detection, market segmentation analysis, computer vision, to name a few. Moreover, machine learning is the central concept used for artificial intelligence and automation development. There are so many other types of algorithms that can be used to variety of problems, multivariate regression, logistic regression, support vector machines, neural networks, PCA analysis, etc., that there is a lot of scope for further learning and implementation.
