Task #4

Regression analysis, gradient descent, multivariate linear regression, polynomial regression and logistic regression

Regression analysis:

Regression analysis is a statistical means for determining the relationship between the dependent variable and the independent variables. It is primarily used for two purposes; it is used for predictions and estimations, having big applications in the field of machine learning and artificial intelligence, and it is also used for defining relationships between the independent and dependent variables.

The most common regression analysis is **linear regression**, in which the best-fit line that most closely fits and maps the dataset is found to define the relationship between the variables. Finding the relationship between income bracket of a citizen to their BMI index is an instance of linear regression. There are also other types of regression analysis, **polynomial regression and logistic regression** for example, that are suited for other situations that a linear regression maybe insufficient for, that will be discussed later in the paper.

Gradient descent:

The most common technique of carrying out regression analysis is the **gradient descent algorithm**. It is an iterative algorithm that aims to minimise the error between the dataset and the initial hypothesis [relationship] that has been defined for the variables in question, by finding the minima of the '**cost function**', that is defined to calculate the error deviation value from the original value in reference to the value that is outputted by the relationship defined. The gradient descent algorithm simultaneously updates the parameters in question, until it reaches **convergence**, that is, the minima, where any subsequent update will have no affect on the parameter values.

Based on the amount of data used, gradient descent algorithm can be further segregated into three types:

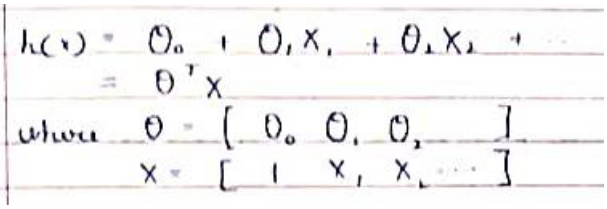
a) **Batch gradient descent**: This implementation of the gradient descent model iterates through the entire dataset to find its minima of the cost function, hence the name 'batch'. This is relatively slower type as compared to the others, as it has to iterate through the entire dataset in one go, but has the advantage of being computationally efficient.
b) **Stochastic gradient descent**: Here, the gradient descent model randomly takes one training example at a time and updates its values. Because of the random nature, it is more likely that this method will find the minima faster than batch gradient descent, as the worst-case scenario is that stochastic gradient descent will have to eventually iterate through all the training examples which would make it equivalent to the batch implementation. One disadvantage is that this method becomes computationally more expensive.
c) **Mini-batch gradient descent**: This implementation is the sweet spot between the previous approaches: the training set is split into smaller 'batches' that is fed to the gradient descent algorithm, thus utilizing the 'random' nature of stochastic gradient descent, but still being relatively computationally effective like batch gradient descent, helping find the minima faster.

As mentioned briefly, the gradient descent algorithm aims to find the minima of the 'cost function' to find the relationship between the variables. The 'cost function', when calculating the error deviation between the hypothesis value and original value uses different methods as well, namely mean error,

mean square error and mean absolute error. The choice of the cost function is not as important to the gradient function itself, but it does amend the minimisation formula to be implemented.

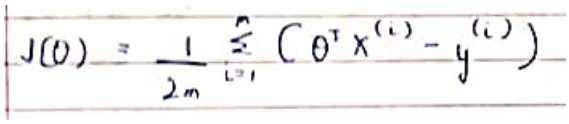
Multivariate linear regression:

Multivariate linear regression is a term used to distinguish the use of linear regression for instances where there is only one independent variable for the dependent variable, as opposed to multiple independent variables for the dependent variables, hence the name 'multivariate' or, 'multiple variable'. The hypothesis function now has multiple distinct variables, named X1, X2 and so on for more readability and convenience in representation in the following formula:

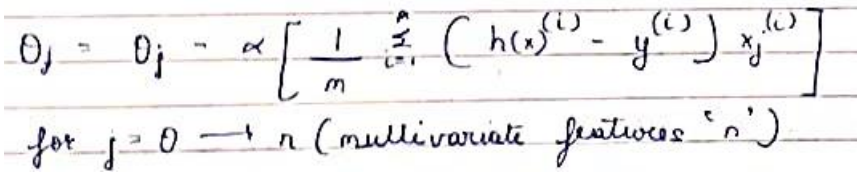


The bias term for the hypothesis function represented by theta0 needs a corresponding X0 term, which will be just 1, for keeping calculations using linear algebra libraries easier.

The cost function too can be represented in terms of the theta vector as shown below:



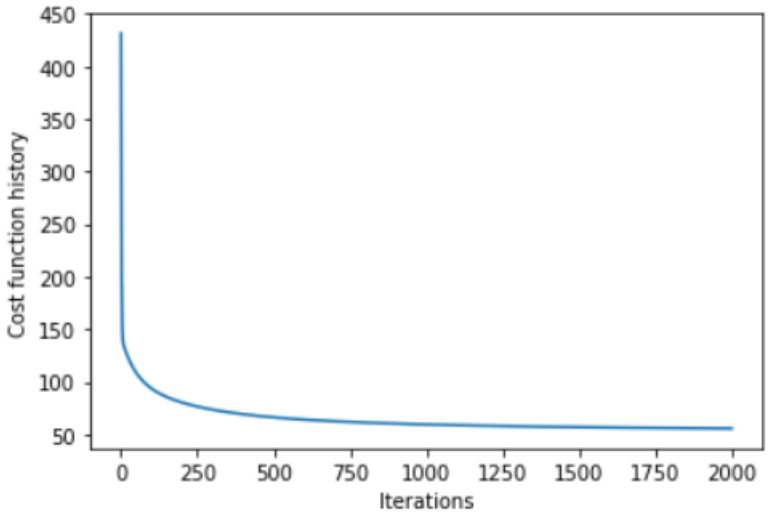
Following is the gradient descent algorithm for multivariate regression:



We can now apply these concepts and see their results. For the purposes of this demonstration, a dataset that has results for concrete compressive strength [in MPa] based on 8 factors, namely amount of cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate [all of these in the unit of kg, for every m^3 mixture] and age [in days] will be used. This dataset has been taken from the University of California, Irvine [UCI] machine learning data repository.

Task #4

A part of the training set will be sliced to save as a test set for evaluating our model later. The code for the cost function and gradient descent is in the GitHub repository for reference. The values of the cost function for each iteration of theta values will be saved to test for convergence. As seen below, the gradient descent algorithm reaches convergence:



The test set is now used to evaluate the training model, by using the theta vector obtained to find prediction values, and the result are as follows:

```
The values for each of the test set examples in an array is:
 [[40.96701712]
 [35.7914805 ]
 [28.22482805]
 [26.8807061 ]
 [33.59457316]]
Comparing the original values in our test set:
 [[44.284354  ]
 [31.1787942 ]
 [23.69660064]
 [32.76803638]
 [32.40123514]]
The standard deviation of the values is: 4.215949613768231
```

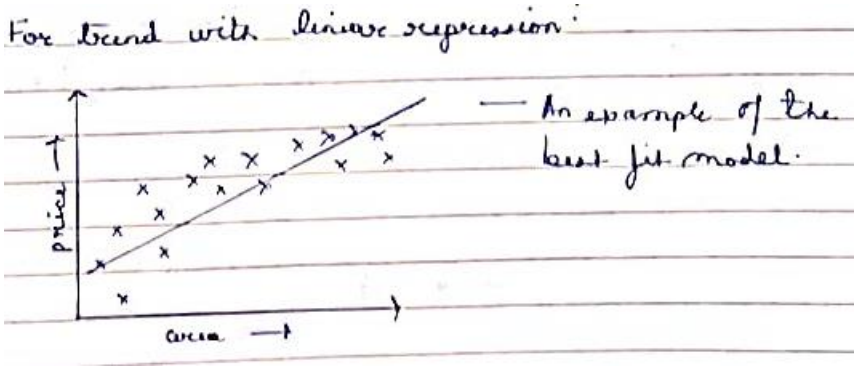
It can be seen that there is some deviation from the original values in the dataset, with standard deviation being approximately 4.22. Through this example, it is illustrated that linear regression is not well suited for such complicated applications the answer obtained after convergence is not as accurate as one might need.

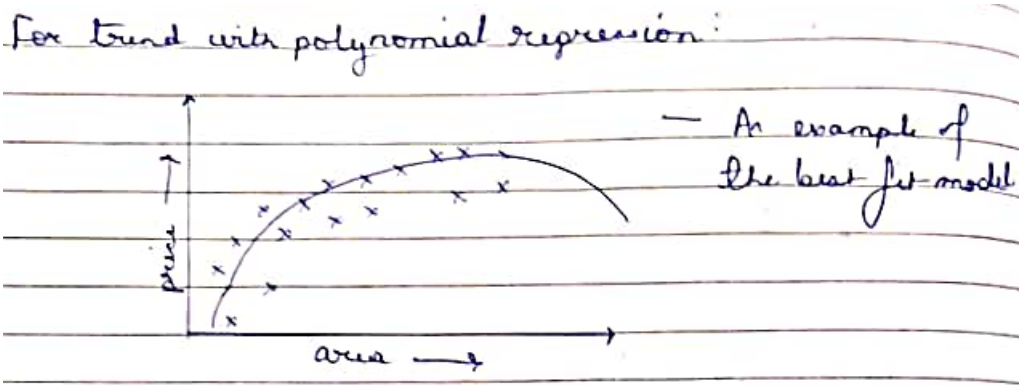
Polynomial regression:

As seen in the previous example, that even after convergence was reached, the theta values could not map the dataset very accurately, having significant error in the values. The solution for more accurate mapping lies in polynomial regression. Here, instead of a straight line, we attempt to fit a curved best-fit line, allowing trends in data to be more flexible and apparent. How we achieve this, is by taking powers of all the features in the dataset and finding the corresponding theta values.

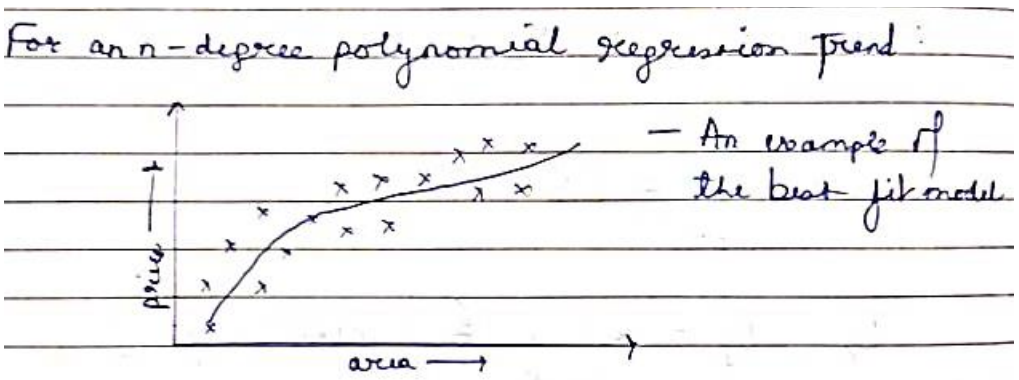
Let's take an example where we can visualise the effect of using polynomial regression as opposed to linear regression. Consider the scenario of calculation of price of house on the basis of one feature,

that is the area of house. Here, we can see that a better fit would be a curved line as the price of the house isn't a strictly linear relationship:



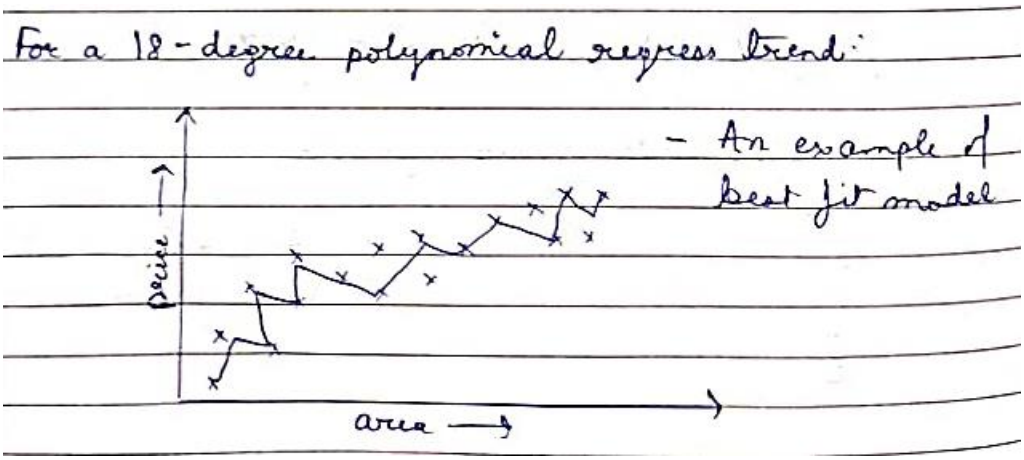


In this example, we used one feature, and a second-degree polynomial. For a larger area, we can observe that the polynomial's trend would then go downwards, which means it is only a good fit for a certain range of values. Therefore, we may need to opt for higher order polynomials or even use the n-root of the features:



Task #4

One factor that comes into consideration is the problem of overfitting. Say we consider an eighteen-degree polynomial, and fit it into our dataset:



The problem here becomes obvious. There is a high bias to the training set values, which means for any area that doesn't belong to the dataset, we may end up with a much more inaccurate value than linear regression itself. The best fit line in this case is overfitting the dataset. When it comes to regression analysis, we look for the more generalised trends, thus, overfitting is a big issue. Therefore, finding the right balance between bias and variance is essential. Moreover, the more squares or roots we take, a greater number of features we have, thus making the training that much more computationally expensive.

Let us implement polynomial regression to our concrete strength problem. Here, we will only take third-degree polynomial of all features to demonstrate the enhanced accuracy through the use of this model. The GitHub repository has all the supporting code. The values for the test set found are as follows:

```
The values for each of the test set examples in an array is:
 [[43.85014869]
 [35.02770197]
 [29.25184778]
 [32.98029434]
 [35.77978533]]
Comparing the original values in our test set:
 [[44.284354  ]
 [31.1787942 ]
 [23.69660064]
 [32.76803638]
 [32.40123514]]
The standard deviation of the values is: 3.3859430262335364
```

We can clearly see that the value prediction is much closer than found in the linear regression implementation, with standard deviation being approximately 3.38, which is a much smaller value than 4.22 than before. By using even high degree polynomials, we can achieve even better results, but consideration to overfitting must be given.

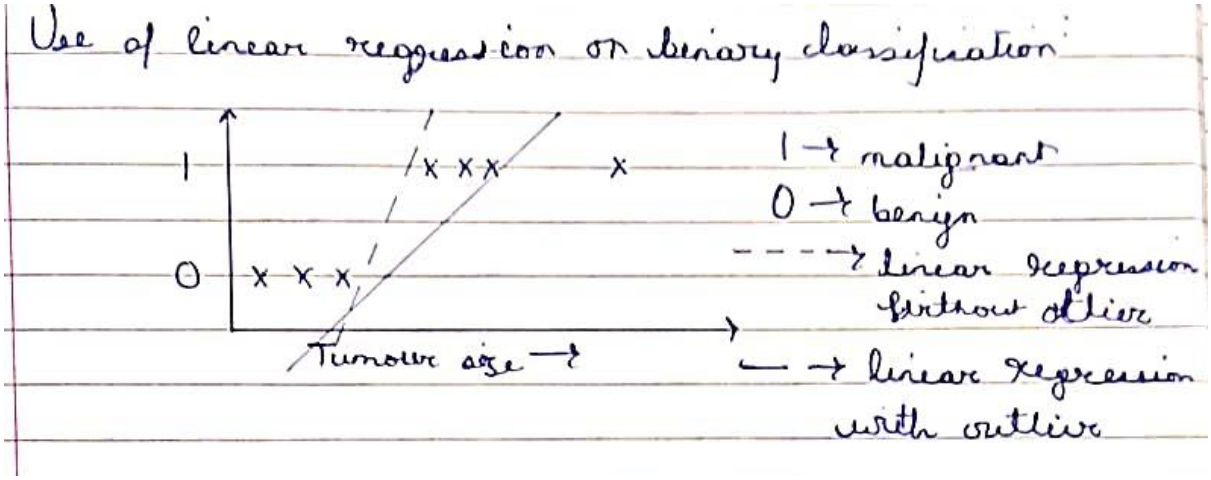
An important note must be given to the technique of feature scaling. When we have features that are numerically very different, such as the squares and cubes that are much larger than our features, the gradient descent algorithm finds it harder to navigate through the data, because of the obvious asymmetry in the value. It becomes more computationally expensive to process the data, and take much larger number of iterations to converge. By feature scaling, we can make every feature

comparable. In this example, the most basic form of feature scaling was implemented, that is, division of every training example with its corresponding feature's maximum value, which converts every value in the dataset between 0 and 1 subsequently. This allows the gradient descent algorithm to be able to update its value faster and find the minima better.

Logistic regression:

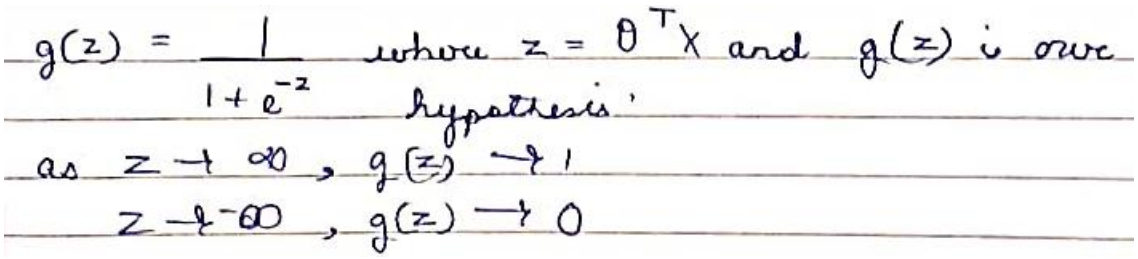
Logistic regression is the regression analysis technique that is suited to classification problems. Here, rather than the general trend in the data and generating a continuous valued output, an answer in classification is required, whether the independent variables suggest whether it is of a particular kind or not.

This can be further understood by an example. Let's consider the simplest example of binary classification: 1 represents a positive class [variables suggest that it is of the type we want] and 0 represents a negative class [variables suggest it is of the type we don't want]. Suppose we want to train a classification model of whether a tumour of a specific size is benign or malignant. To create our hypothesis, if we use linear regression to solve this problem, our output will look something like this:



For simple cases, the linear regression could work fine. However, in the case of an extreme outlier as seen in the figure above, the linear regression hypothesis falls shorts of being adequate. What we need is a classification boundary to be able to create an accurate hypothesis.

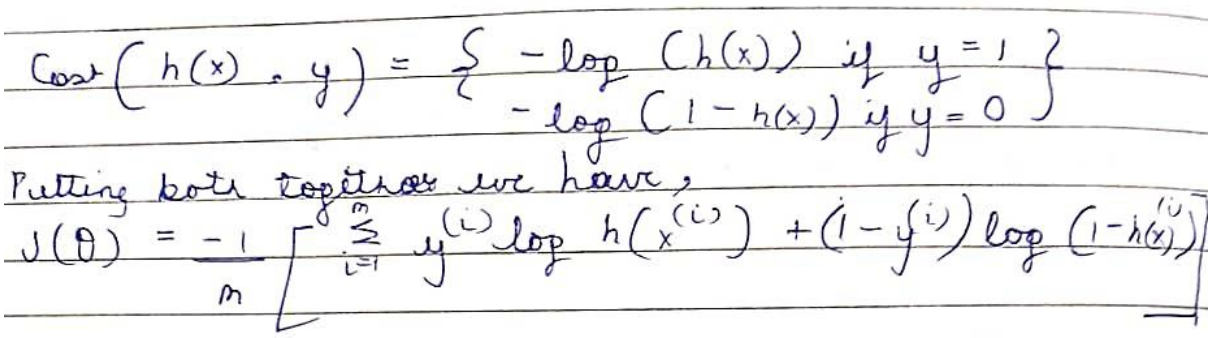
For the creation of the classification boundary, we need our hypothesis function to output a value between 0 and 1, so that we can define true as any value that is greater than or equal to 0.5, anything lower is false. For this purpose, we make use of the sigmoid function:



Thus, for given size of tumour, we have the 'degree of truth' of that tumour size being malignant.

Task #4

For logistic regression, the cost function too changes. The previous cost function for linear regression cannot be used because it will yield a non-convex function and thus there will be too many local minima that can be converged to, so, we define a new cost function:



The equation for gradient descent remains the same in notation, however, h(x) is now completely different from linear regression.

We will consider a dataset from 'Machine learning by Stanford' course on Coursera, with the problem statement of determining a student's chance of admission based on two test scores. The supporting code can be found in the GitHub repository. After implementing the model, we get our results for the test set as follows:

```
The values for each of the test set examples in an array is:
 [[0.66179484]
 [0.64010208]
 [0.69922627]
 [0.63831514]
 [0.68523785]]
Comparing the original values in our test set:
 [[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

We observe that all our values for answers come above the 0.5 threshold, which means that all our answers our correct. Thus, we have created a decision boundary that functions well the model.

Conclusion:

Regression analysis is thus a powerful means of obtaining the relationship between independent variables and dependent variables. However, there are a lot of way to implement the models, and as we have seen with various examples, each of them is better suited for a very specific scenario. It is thus a good practice to review what the demand of the problem statement is beforehand, prior to the implementation of the model.