**Abstract**

This project explores some standard machine learning techniques to build a classifier - with the intention of refining them to improve the results of their classification.

Two such techniques are explored: Logistic Regression and Deep Neural Networks. We use the same data for every algorithm in order to get a better comparison between their performances.

In each case, we start with simple algorithms and then explore some variations in each of them. The results of each variation are recorded, in order to compare between them and establish which gives the best performance.

Throughout this project, we use the Iris Flower Data Set for every classification - a standard dataset that is used commonly in data analysis.

**Machine Learning**

Machine Learning is the use of statistical techniques to make computers ‘learn’ with data, without being explicitly programmed. It is a subset of Artificial Intelligence in computer science. Arthur Samuel coined the term machine learning, and the field evolved from studies of pattern recognition in the late 1950s.

It involves the construction of algorithms that can learn from and make predictions on data – instead of following static programmed instructions. This is done by building a model from sample inputs through various learning algorithms, such as logistic regression or neural networks. It can solve problems whose solutions are difficult, if not infeasible, to explicitly program algorithmically – for example: Optical Character Recognition, Email spam filtering and computer vision.

Machine Learning is mainly concerned with 4 learning tasks –

* Supervised leaning: The computer is given example inputs (called data) and their known outputs (also called labels) and the objective is to learn a rule that maps each input to the given output.
* Unsupervised Learning: No target labels are given to the program, instead only the input data is presented. The goal of the program is to find some structure in the given data. You’re basically saying to the computer: ‘Here, take some data and make sense out of it’.
* Semi-supervised Learning: We give incompletely labelled data to the program. Often, most target labels are missing and less than 10% of the data are labelled.
* Reinforcement Learning: The program is asked to make decisions in a dynamic environment and the training data consists of rewards or punishments given as a consequence of the action the program took. For example, teaching a computer to play a video game.

**Classification**

Classification is an application of machine learning where all the labels belong to a finite set of values that is known to the program. As such, classification falls squarely in the category of supervised learning. For eg: classifying emails as spam or not spam.

In classifications, the labels can be represented by integers starting from zero, by binary digits or by a vector of one-hot coded values.

Often in classification tasks, we are concerned only with classifying objects (data) into either of two categories – this is called Binary Classification.

Besides classification, two other common applications of machine learning are:

* Regression, where the output variable (ie the label) belongs to a range of continuous values instead of a discrete set
* Clustering, where a set of inputs should be divided into groups and the set of groups is not known beforehand.

**Data Set Description**

The Iris flower data set is a multivariate data set introduced by statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems*. It was collected by Edgar Anderson in order to quantify the morphologic variation of Iris flowers of three closely related species.

The data set consists of 50 samples from each of the species *Iris setosa*, *Iris virginica* and *Iris versicolor.* Each sample/record contains measurements of 4 attributes of the flower-

* + - * Petal length
      * Petal Width
      * Sepal Length and
      * Sepal Width

The data is stored and read from a CSV file where each row contains 1 record followed by the name of that species, ie, data followed by label.

It has been shown by previous analysis that out of the 3 species, Iris setosa is fully linearly separable from the other 2 species whereas Iris virginica and Iris versicolor are not completely linearly separable, although it is only a few examples that make the separation non-linear. Hence, with this dataset, we can investigate both the performance on linearly separable data and non-separable data for any given model.

**PART I – The Logistic Regression Approach**

**Intro**

Logistic regression is a widely used classification technique in Machine Learning. It has a relatively modest number of parameters – just one parameter for every input feature – making for a very simple model; it is not suited to complex classification tasks. However, by using a simple model, we can eliminate the risk of overfitting the data and save processing and computation time in training our model.

In the logistic model, we group each set of inputs into an input vector and further gather all the parameters of the model into another parameter vector. We begin by taking the dot product of the input vector and the parameter vector. This dot product is passed through the logistic function to get the final output of the model – always a real number between 0 and 1. A mathematical explanation can be found in [[insert section number]].

[[[Formula of Logistic **Function** is to be put here]]]

**The One v Rest Approach to Classification**

Although the data we have chosen contains three classes of inputs, the logistic regression model is best suited for binary classification problems. Hence, we need to convert a ternary classification problem into this form. This is achieved using one-vs-rest classification, where we classify each example as belonging to one particular class (label ‘ONE’) vs not belonging to that class (label ‘REST’). This class needs to be chosen beforehand and typically, the choice of this class influences the ease of classification. The OnevRest approach can be used to easily convert an n-class problem to a 2-class one, so that Logistic Regression becomes possible.

Further, OnevRest approach also suggests a method to run logistic regression on a fully multi-class problem. This method is commonly used for the logistic technique; however, it will not be explored in this project. If there are C classes, we would design C classifiers, each of which determines whether the input example belongs to one of the C classes or not, a la OnevRest. Then, by comparing the classification scores of the C classifiers, we select the best class that the example would fall into. From previous analysis of the Iris Data Set

Since the Iris Dataset has three classes of inputs, there are three ways to implement One v Rest classification – each one classifying a specific class vs the rest.

**Mathematical Description of Logistic Regression**

[[ Put symbol of theta wherever **Th** is there ]]

**X** = [x1, x2, … xn], are the inputs to Logistic Regression

**X** = [x0=1 : **X**] : x0, which is always 1, is appended to the inputs from each example in order to match the length of the parameter vector **Th**.

**Th =** [th0, th1, …thn]: these are the parameters

**Z** = **X.Th**

**H** = logit(**Z**)

**Training the model**

**What is Training?**

In classification, the function of any model is to take a set of inputs and predict some output values that decide which class the input example belongs to. In binary classification though, it predicts just a single output, typically between 0 and 1.

In order to accomplish this, the model has some parameters that the input is computed with. Thus, the value of the parameters defines wholly what the output is going to be for every input shown to the model. The task of training (called fitting the model) is to tune these parameters such that the model can accurately distinguish between inputs of different classes. Thus, we take some data for which the class labels are known and then determine the model’s predicted value for each input example. Next, we choose an appropriate cost function

**Cost Function**

The Cost Function measures the error in classifying using a given model. The more the predictions correlate with the actual labels, the lesser should be the value of the cost function. Thus, the task of fitting our model accurately is accomplished by monitoring the cost function and reducing it’s value by parameter tuning. Hence, the cost function should also suggest which way to tune each of the parameters of our model.

For logistic regression, we use the following cost function:

J(Hth(x), y) =

* -log(Hth(x)), if y = 1
* -log(1 - Hth(x)), if y = 0

Since the behavior of the function is defined piecewise, its shape is as shown in fig [[insert fig number]]. Notice that its tendency is towards zero as the prediction tends toward 1, in case a where the true label is 1, or as the prediction tends toward 0, in case b where the true label is 0.

[[[ Put both pieces of logistic’s diagrams here and label them case a and case b as per the above sentence ]]]

To accomplish the tuning of our model’s parameters in such a way as to minimize our cost function, we need to choose an appropriate Optimization Algorithm.

**Gradient Descent – The Cost Minimization Algorithm**

Gradient Descent is a first-order iterative minimization algorithm for finding the minimum of a function. It is based on the following observation: *That if a multivariate function F(****X****) is defined and differentiable in the neighbourhood of a point a, then it decreases fastest in the direction opposite the gradient of F at a, or* [[-del F(a)]].

Thus, in every step, we move a distance proportional to the local gradient, in the direction of the local gradient. This gives us the following update step: <https://wikimedia.org/api/rest_v1/media/math/render/svg/26a319f33db70a80f8c5373f4348a198a202056c>

[[alpha]] here is called the learning rate and determines how much we move in every step and, thereby, how fast we will converge.

Since we are guaranteed to move down at every step, and since we would stop only if the gradient became zero, we can expect to converge on a local minimum of the function reasonably quickly. The two main caveats of this approach are that:

* A) We cannot ensure that the minimum we find will be the global minimum
* B) If the learning rate [[alpha]] is too large, an update might lead to a cost increase.

**Gradient Descent Applied to Logistic Regression**

The cost function, J(**Th**) from [[section no.]] is rewritten as:

J(th) = -y\*log(Hth(X)) – (1-y)\*log(1-Hth(X))

So as to be readily differentiated. Note that it is written as a function of our model’s parameters Theta because we are trying to minimize the cost function by varying those parameters only.

[[ Put the math here, of the steps of finding the gradient of J w.r.t **Th** - conclude with the UPDATE STEP]]

**Experiment Ia**

**Procedure**

Three classifiers were trained to distinguish (one each) between Iris setosa and the rest, Iris virginica and the rest, Iris versicolor and the rest. The model parameters were initialized to all 0’s and trained for 10k iterations using Logistic Regression. The cost was monitored after every iteration for later analysis. The learning rate was chosen to be [[alpha = 0.1]]. Only 80% of the dataset (40 per class x 3 classes = 120 examples) was used for training so that the remaining 20% of the dataset (10 exmaples per class x 3 classes = 30 examples) could be used to test the trained model. The graph of Cost vs No. of examples was plotted for the first 1k iterations for each classifier. *(The trend in the graph continues for the remaining 9k iterations as well)*

**Results Obtained and Analysis**

1. **Classifier 1 – Iris setosa vs the rest**

From Training: Initial Cost = 66.233; Final Cost = 0.117

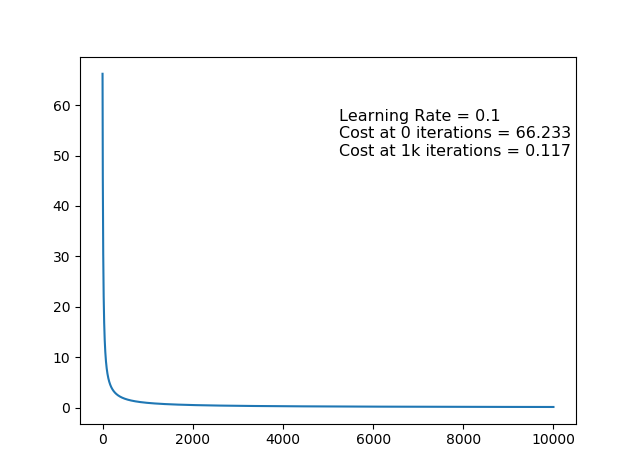
From Testing: Correctly Classified: 30 examples (100%)

Incorrectly Classified: 0 examples (0%)

Confusion Matrix [10, 0]

[0, 20]

Graph of first 1k iterations of training



**Observations Drawn**

* The cost reduces sharply for the first few iterations but slowly for later iterations.
* The Cost plateaus out after a few hundred iterations, and it does not tend to zero.
* The Cost consistently falls after every iteration, although the fall is less later on.

1. **Classifier 2 – Iris virginica vs the rest**

From Training: Initial Cost = 78.147; Final Cost = 8.77

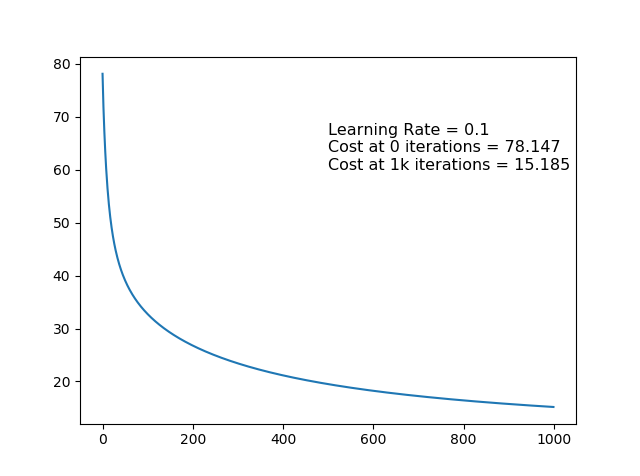
From Testing: Correctly Classified: 29 examples (97%)

Incorrectly Classified: 1 examples (3%)

Confusion Matrix [10, 0]

[1, 19]

Graph of first 1k iterations of training



**Observations Drawn**

* The cost falls quickly for the first iterations, but not as for 1. Iris setosa
* The Cost plateaus out after a few hundred iterations, and it tends to a value higher than for 1. Iris setosa
* The Cost consistently falls after every iteration, although the fall is less later on.

1. **Classifier 3 – Iris versicolor vs the rest**

From Training: Initial Cost = 78.278; Final Cost = 63.2

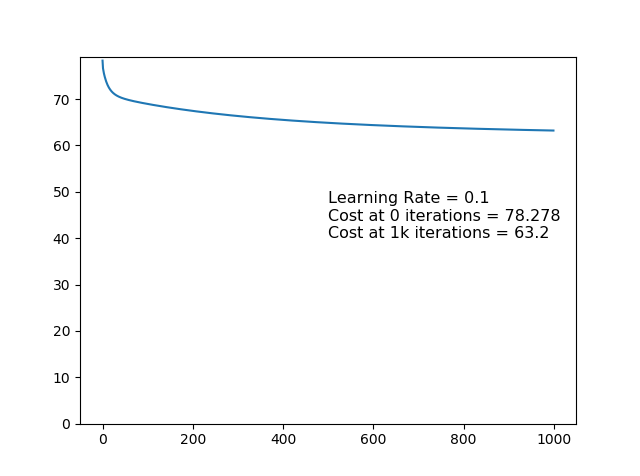
From Testing: Correctly Classified: 22 examples (73%)

Incorrectly Classified: 8 examples (27%)

Confusion Matrix [3, 7]

[1, 19]

Graph of first 1k iterations of training



**Observations Drawn**

* The cost reduces appreciably only for a few starting iterations.
* The cost plateaus out far earlier than for 1. Iris setosa or 2. Iris virginica
* The total reduction in cost is minimal overall and does not tend to improve with more iterations.

**Experiment Ib**

**Ideation**

* In Gradient Descent, the size of the update step is proportional to the magnitude of the gradient of the Cost Function [[del theta = -alpha\*grad(Th): quote this from earlier section here]].
* Therefore, as the magnitude of Cost and that of it’s gradient fell in the previous experiment, the size of the update step became smaller and smaller until finally, towards the end of the training, the update step size was negligible.
* How can this effect be counteracted? How to ensure that the update step size does not become negligible?

**Procedure**

For the following experiment, the learning rate was made adjustable. The value of [[alpha]] was increased as the value of Cost fell, in three different ways. The point of interest was how far and how fast the cost would fall as compared to Experiment 1a, where [[alpha]] was a constant.

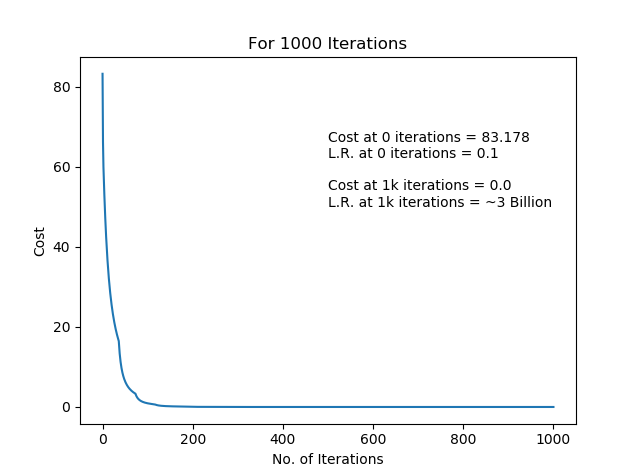
1. Every time the Cost fell by a factor of 10, [[alpha]] was increased by a factor of 10. **Update Step:**

The following were the observed values:

1. Every time the Cost fell by a factor of 5, [[alpha]] was increased by a factor of 5.

**Update Step:**

The following were the observed values:



1. Every time the Cost fell by a factor of 2, [[alpha]] was increased by a factor of 2.

**Update Step:**

The following were the observed values:

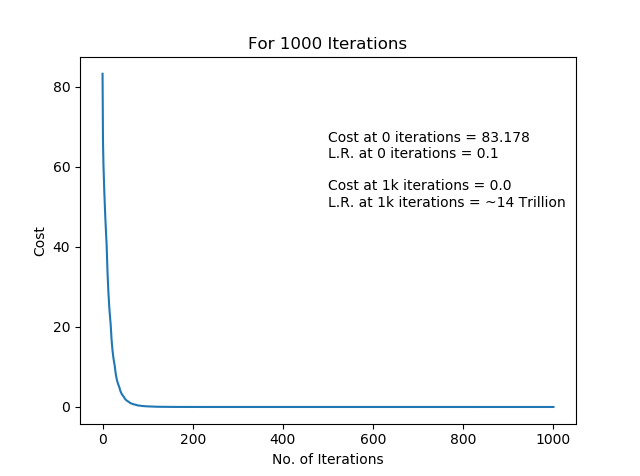


Table 1 [[renumber it]]

|  |  |  |
| --- | --- | --- |
| Scale Factor | [[alpha]] after 1000 iter. | Cost after 1000 iter. |
| 10 |  |  |
| 5 |  |  |
| 2 | 14 Trillion | 0.0 |

Table 2 [[renumber it]]

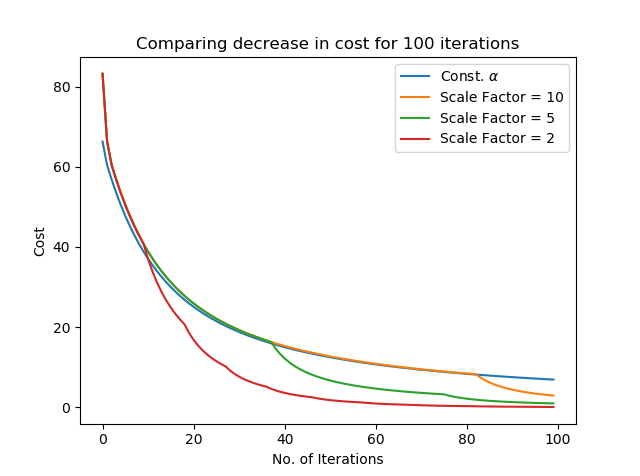
|  |  |  |
| --- | --- | --- |
| Scale Factor | [[alpha]] after 100 iter. | Cost after 100 iter. |
| 1 | 0.1 | 6.88 |
| 10 | 1.0 | 3.002 |
| 5 | 2.5 | 1.007 |
| 2 | 51.2 | 0.142 |

Table 3 [[renumber it]]

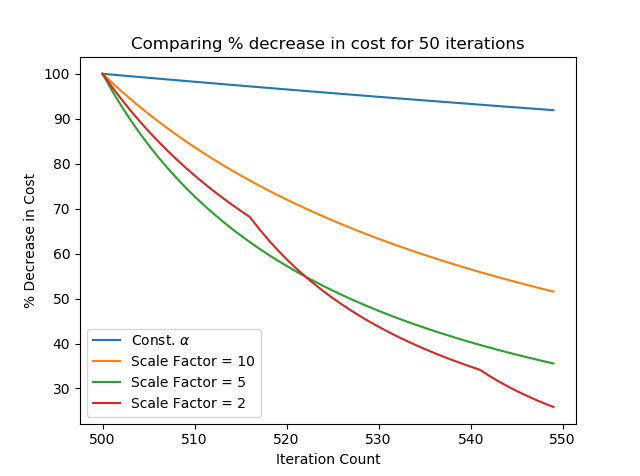
|  |  |  |
| --- | --- | --- |
| Scale Factor | No of iterations till Cost < 0.1 | No of iterations till Cost < 0.01 |
| 86 | N.A | N.A |
| 10 | 286 | 444 |
| 5 | 178 | 270 |
| 2 | 103 | 162 |

*[With the standard logistic algorithm, the cost didn’t reach 0.1 even after 10k iterations]*

The graph of Cost vs No. of Iterations was plotted for the first 100 iterations to compare the standard Logistic algorithm with the 3 versions proposed here.



Since the intent behind this experiment was to negate the update step becoming negligible in the later stages, the following graph was plotted which depicts the decrease in cost for 50 update steps after 500 iterations have been completed, comparing these values for the original Logistic algorithm and the 3 others proposed here.



**Conclusions Drawn**

* In all cases, the cost fell faster than in Experiment Ia, where [[alpha]] was a constant. Hence, these changes have led to an improved Logistic Regression algorithm, as was expected in the Ideation section of this experiment.
* Introducing any scaling factor leads to dramatic improvements over the standard Logistic algorithm.
* From graph [[graph number - 100 iter graph]], the smaller the value of the Scale Factor, the better the performance of the algorithm.
* From graph [[g. no. – 50 iter in the middle graph]], there were dramatic improvements to the algorithms performance in later stages of training. Lower S.F performed better here.

**Part II - Neural Nets**

**References:-**

https://en.wikipedia.org/wiki/Machine\_learnin