**Problem Statement**

In this assignment we will analyze algorithms for discriminative learning using logistic regression and neural networks. The report has two main parts:

* The logistic regression
* The multilayer perceptron

For logistic regression we will use the Iris dataset, and for the multilayer perceptron we will use a dataset of images gotten from the datasets of sklearn. In addition, for both datasets we will use cross validation to evaluate the performance.

In order to make it easy to understand, each experiment has its own source file and dataset file.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 |
| Source File | literalA.py | literalB.py | literalC.py | literal2B.py |
| Dataset | iris1.dat | Iris1.dat | iris.dat | sklearn dataset |

To run any of the experiments, it is just required to execute the corresponding python file.

In addition, we created a utilities file that includes some methods that are useful for almost all of the experiments, such as the calculation of the precision, recall, accuracy and f-measure. This file is imported in each of the experiments, so the methods could be used freely.

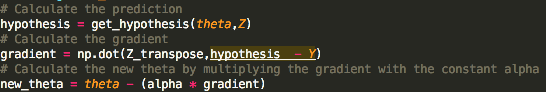
**Logistic Regression**

For these experiments we will use the Iris dataset with two classes and three classes. In the experiments one and two we will remove one class in order to fulfill our requirement.

Then we proceed with cross validation with k = 10 to evaluate the performance of our algorithm.

**Experiment 1**

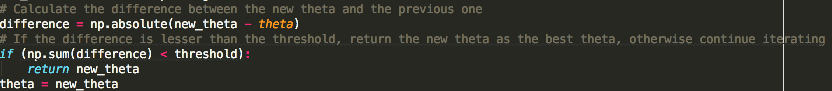
The first part in our algorithm is creating a theta zero with random values from 0 to 1. With this we start calculating the gradient and the new values for theta zero.



The hypothesis calculated here is the sigmoid.



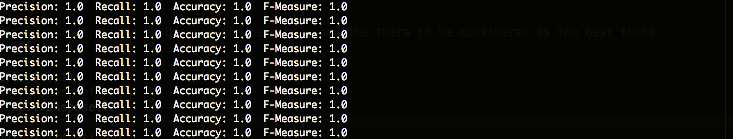
Then with the parameters learning rate set to 0.0005 and the threshold for the difference between the thetas to stop the condition as 0.0001, we find the theta for our algorithm.



We continue until we find the new theta or a maximum number of iterations to prevent an infinite cycle.

All of the experiments are implemented in python, and all of the operations are done working with matrices in order to make it faster and avoid the slow python loops.

At the end we get the error shown below:

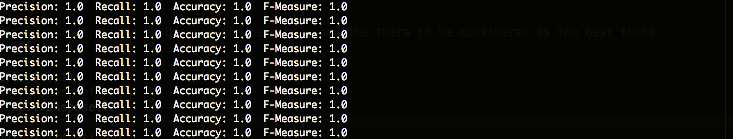


We can see that our classifier did a perfect work. This is understandable because the Iris dataset is a simple dataset that contains a big gap among its labels, making it easy to identify and distinguish them.

**Experiment 2**

Like experiment 1, we set an initial theta vector with values from 0 to 1 and we calculate the gradient but with a higher dimension. In this case we increase the dimension to degree 2 and then do exactly the same process as we did in the experiment 1 by calculating the hypothesis with sigmoid, calculating the gradient, and stopping the process when we find a theta that does not change a lot. This difference defined by a threshold of 0.0001.

At the end we get the error shown below:

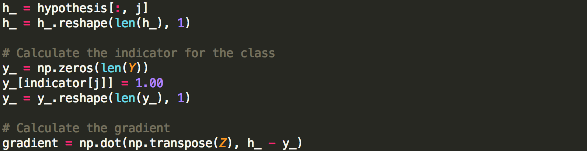


Once again we can see that our classifier did a great job. In this case it was not required to increase the dimension of our features because by having the 4 defined by default, was more than enough to find a good classifier. However, this does not happen always, and this is the reason why we need to try with different dimensions and algorithms to find a good classifier.

**Experiment 3**

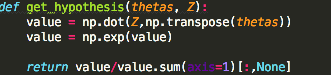
In this experiment we use the Iris dataset but with three classes.

The first step was to define a matrix of thetas, one for each class. The values for the thetas were chosen as 0.01. Then with this we move to calculate the gradient.



We do the operation as matrices. So here, we take the column of the hypothesis that belong to the class j. Then we subtract the indicator of Y and finally we calculate the dot product between them.

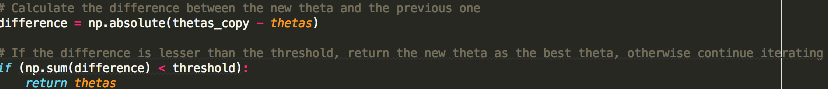
To calculate the hypothesis we use the softmax.



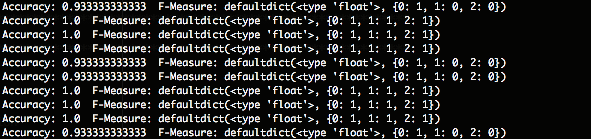
After that, by defining a threshold of 0.0001 and a learning rate of 0.0005, we calculate the new thetas.



Finally to see if we find the right thetas, we calculate the difference between the previous theta and the new one calculated. If the difference is less that the threshold, we stop the loop and return the thetas found.



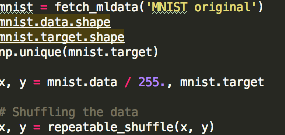
At the end, to predict we use the same hypothesis but with the X values of the ones we want to predict. With this we get the error below:



We can see that the errors are small. This means our classifier is doing a great job. However, we could figure out that the ones from the previous experiments did it much better.

**Experiment 4**

In this experiment we will use an image dataset from sklearn. So by that, we start reading the dataset and shuffling the data.



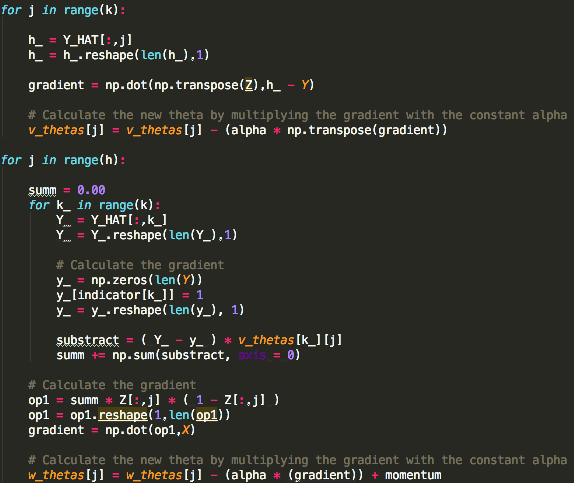
With that, we can start defining our v and w with values of 0.01.



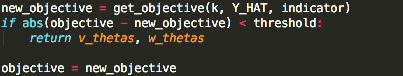
In this part we define h as the number of features we will have in Z as the output of the hidden layer. Here we will define three values for h: a lesser value, the same as the number of the original feature and a higher value. Then we calculate the hypothesis. First, to calculate Z with x and w, we use the sigmoid, and then to calculate Y\_hat we use the softmax.



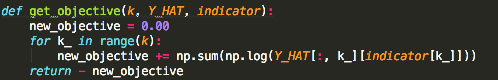
Once we have this, we calculate the gradient for V and W.



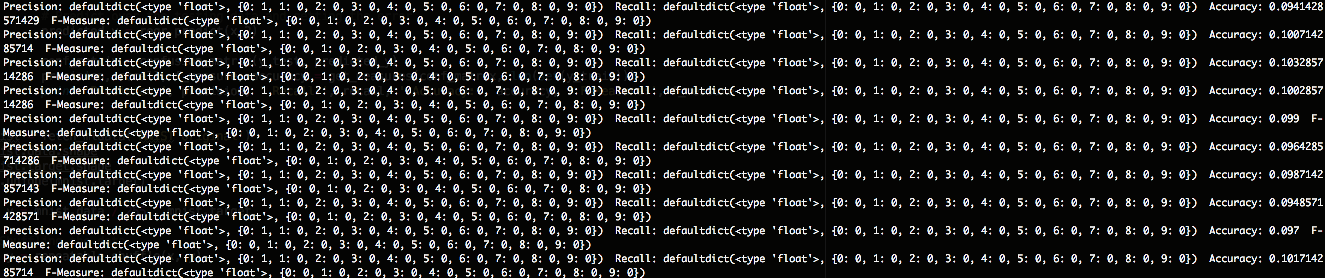
In order to calculate them both, we use the learning rate of 0.0005 and a threshold of 0.0001. In addition to calculate W we also use a momentum of 0.05. We continue searching of the values of V and W until our objective does not change a lot.



Our objective is defined as the maximum likelihood of thetas. So to calculate it we use the code:



At the end we get the error below:



We can see that our error is high which means that at least for this dataset, our classifier is not doing a great job. By increasing the dimensions or decreasing the dimensions, we always get a high error.

**Exercise 2A**

