EDAP01 Lab 3

Kalle Josefsson ka5532jo-s

September 17, 2024

1 Introduction

The purpose of the lab was to try and classify if a text was written in French or English using machine learning in different ways. The data used was a book called Salambo which we got both a copy in English and French. The book is divided into 15 characters and for each chapter the amount of characters are counted as well as the counts of the character A. This was done for each book, hence we had thirty triplets of total characters, how many of them were A's and a one or zero to classify the language the texts was written in. This was the dataset which we used for training for each model we created. After training we tested our model and here I will present and analyze them.

One note is that all figures and results that are discussed in the report is found in the notebook that will be submitted seperately with the report.

2 Implementation of classifiers

2.1 Gradient Descent method to implement linear regression

The first step in the process of the implementation was to split the dataset into an X-matrix which is the input and the y-vector which is the target. I then normalized them, I choose the range zero to 1 since we are working with classifiers I think its fitting to keep the values between zero and one because then you can look at them as probabilities in a way it also keeps makes it also decreases the computational power needed as well as removes risk of failure due to descent algorithms being sensitive to the range of the dataset. The next step is to implement the descent functions for the weight updates. The idea here is to minimize the error by updating the weights after each epoch based on the error, and when the gradient is small enough we are satisfied as it is seen as a minima and if we do not reach the minima we stop after a predefined number of epochs. This was done by both standard gradient descent, i.e using the entire dataset to update the weights for each epoch and using stochastic gradient descent which we look at a random part of the dataset and update the weights based on the error from these data points. After training the descent models we got a set of weights for each descent model we then plotted the lines with our data points to see how well we fitted the model to our data set. Luckily both for the normal batch descent and then stochastic descent our model seems to fit very well to the data points. Note that we had to reverse the values back to their actual values from the normalized ones which was easily done, we also had to change the weights accordingly. The linear regression was only done for one of the languages and I choose to do it for the English dataset.

Now to the classification part. In order to classify we need to use both the French and English dataset of course. We normalize them again and plot a scatter plot of the data points. As we can see in the graph there is a possibility to linearly separate them and then we have a linear classifier. For this we used the percetpron model which is discussed below.

2.2 The perceptron

To create our perception we introduced a prediction function which does exactly that, it returns a value of zero or one based on which language it thinks it is. Based on the the matrix multiplication between the weights and input we get our values and based on the values obtained we have a threshold being zero, if the y-value for a data point and weight set is larger than the threshold value we classify it as a

one otherwise zero. Now that we have a predict function we can create our descent function. I choose to do the stochastic descent for all of the remaining tasks we also used a learning rate of alpha equal to one here as it worked well in the previous stochastic descent model. This descent worked similarly to the previous descent functions except that we stopped training not when the gradient reached a minima but when the classifier made the correct prediction for every data point during one epoch. After training we plotted how many epochs it took to reach zero classifications and the weights. The average amount of epochs needed was around 30 which I was rather satisfied with. Then we plotted our classifying line which seemed to separate the blue and red dots which meant a useful model.

Now to the real evaluation of our model we had to test it on data which was not seen in the training but which is from the same distribution. A smart way to do this is to leave one of the data points out from the training and then test our model on that point. I did this using all 30 data points as test data one time and noted the result. This obtained an accuracy of 1.0, i.e it classified all the data points correctly. One important note is that the amount of epochs needed to obtain zero missclassifcations varied a lot and ranged between two and 125.

2.3 Logistic regression

Now to the last implementation we had to do during the lab. This was to implement logistic regression. In order to do this I implemented three help functions. The first one was just the logistic function which just returns the logistic vector or value for a vector or value. Then I made a predict probabilities function which calculated the values for y using the same matrix multiplication which we used in the other models and then put this value in the logistics function gave it a value between zero and one. The last help function was the predict function which based on the predicted value made a prediction either zero or one based on if the logistic function gave a value higher or lower than point five. Again we update the weights based on if the prediction is correct and the change is based on how far off our prediction is from the actual value. This time we early stop if the norm of the change in weights is less than a predefined constant epsilon hence the model thinks that the weights are good and stops updating them (almost) and therefor I early stop this took on average a little longer than stopping after zero misclassifications during one epoch. Now to the results from this model was similar to the last task, the accuracy was again 1.0 but number of epochs to reach was higher and the computation time longer.

2.4 Comparing with popular tools

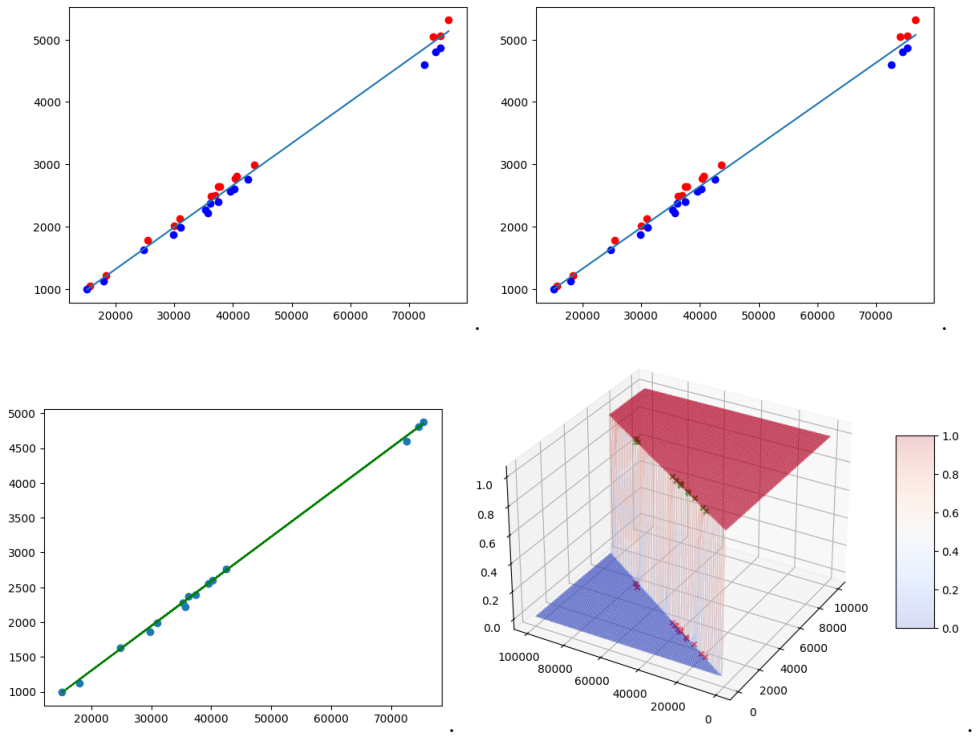
I was supposed to compare the results from my implementation to some popular machine learning tools. The tools I was supposed to compare with are Keras, Pytorch and sklearn. All of the three different tools solved the classification problem getting an accuracy of 1.0 across the board. sklearn did work the best however since it was a lot more "sure" than the other to models. When the correct classification was one it got a probability for the class being zero to ten to the power of minus 30, i.e really small numbers while Keras and Pytorch got probabilities that were a lot closer to 0.5 which means that model is fairly uncertain of what the correct answer is. Hence the sklearn model would probably work a lot better on new test data.

2.5 Results

Task/Weight	w1	w2	w3
BGD	0	-3.56432855 7	0.06430049
SGD	0	16.51596702	0.06423801
Perc	0	-0.2665553552107331	4.05835843373494
Log Reg	2.1123451523439485	-0.004125347154952724	0.0609967407551438

Table 1: Weights for tasks

The first two plots shows the linear classifier between the two languages and the the bottom left shows the linear fit that I got for describing the amount of characters based on the amount of A's in an English text.



2.6 Discussion of article

The last task of the lab was to read the article "An overview of gradient descent optimization algorithms" by Ruder (2017). Then I am supposed to outline the main characteristics of each optimization algorithm which I have done below.

The first algorithm discussed is Momentum, which handles ravines which are parts of the surface where the gradient in one dimension is much larger than in another this makes normal gradient descent uncertain and makes it oscillate while if we add a momentum vector, i.e a vector from the past update which accelerates convergence since it pushes forward in the direction we took the last time step. This also reduces updates for dimensions when the gradients switch signs repeatedly and therefor dampens oscillations.

The Nesterov accelerated gradient (NAG) improves the normal momentum method discussed above. It does this by calculating the gradient of the loss function at approximate future positions. This allows the NAG to update more accurately instead of just sprinting towards the minimum without a plan which momentum can cause.

The Adagrad algorithm does not have a constant learning rate but changes it based on parameters. It has different learning rates for each time step we take and for each parameter we have. one very positive outcome from this is that we do not have to spend time finding a learning rate which fits the model since this does it for us. Also if the gradients are very different in different directions of course want different learning rates. But it has the weakness that as we add more values the learning rates shrink and eventually become so small that wee can not use new data points and therefor the model stops learning.

Adadelta is the algorithm that counters the weakness in Adagrad and is an evolved version of it. Instead of taking all the previous gradients into consideration when setting a learning rate for a parameter it just looks a fixed time back hence neglecting the risk of vanishing learning rates. ANother positive outcome is that since the learning rate is based on the previous ones we do not even have to set an initial one.

The next algorithm discussed is RMSprop which purpose is do the the same thing as Adadelta does, i.e

prevent vanishing learning rates in Adagrad however here we still need a default learning rate making it more sensitive to human error when we choose hyperparamters.

Adaptive Movement Estimation (ADAM) also computes learning rates at each timestep for each parameter and also stores decaying average past squared gradients but also does the same thing for second moments of of the gradients like momentum. This gives a better balance between momentum and adaptive learning.

AdaMax is a variant of ADAM but has better capability to handle unstableness by using the infinity norm instead of using a higher number than l1 and l2 which are normally stable. This due to the fact that the infinity norm is less sensitive to large gradients.

The final algorithm discussed was the Nadam algorithm combines the ADAM optimization algorithm with Nesterov momentum. The combination is made by adding the lookahead feature in NAG into ADAM's framework. This gives a more refined updating strategy