Project 3 EDAP01

Sophia Kovalenko [so4816ko-s]

March 2024

1 Presentation of the assignment

In this assignment, the goal was to practice the theory about linear classifiers: regression and classification and applying it on data from the book Salammbô. The objectives consisted of writing a linear regression program using gradient descent to predict the counts of A's in a text from the total count of letters.
Then, writing linear classifiers using the perceptron algorithm and logistic regression to classify a chapter as French or English. After evaluating those implementations, one had to read a scientific article by Ruder on optimisation algorithms.
In this report, I will present my implementations as well as their results on the datasets given. Finally, I will discuss the scientific article aforementioned.

1.1 Possible improvements

One improvement I would have brought would be to name the methods differently for each algorithm. Jupyter notebook performs well in projects like those, if and only if one runs the whole notebook each time. As when running each cell, it overwrites methods with the same name which were ran previously. Which can cause some undefined behaviour when implementing and testing separately.

2 Presentation of my implementation

Gradient descent minimises a function by updating its parameters in the opposite direction of its gradient. When it reaches 0, the algorithm has found a minimum.
In the implementation, there are two descent methods: batch and stochastic gradient descent. The difference between them is the way that the weights are updated. Batch descent updates them using the whole dataset whereas stochastic descent updates them after each observation.

2.1 Regression - Linear Regression

In this part, I used gradient descent to estimate weights of a function to fit the dataset provided. First, I extracted the feature matrix X also called predictors

which corresponds to the letter frequencies. Then, I extracted the response: the class vector y which represents the frequencies of the letter A. After normalisation, I fit the data using first batch descent and a stochastic fit in another run. The stopping criteria is either the maximal number of epochs, or a small enough gradient ($\|gradient\| <$ epsilon).

This next part's goal is to classify the points into French or English chapter using two different algorithms. Given a pair of numbers corresponding to the letter count and count of A, it will predict the language by finding a linear decision boundary. In Figure 1 one can visualise both of the datasets. Here I used stochastic gradient descent as it performs better with big datasets. The reason is that it does not perform any redundant computation like the batch gradient descent does.

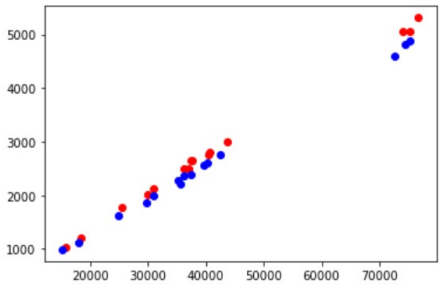


Figure 1: Datasets, red: French, blue: English

2.2 Classification - Perceptron

The perceptron works by finding the weights for the separating hyperplane. It iterates until either the maximum number of epochs, or until all the points are correctly classified. The loss only updates for the misclassified points.
The perceptron consists of two main functions: predict(X,w) and fit(X,y).
The predict function uses the hyperplane to predict the output $\hat{y}$, vector classes 0 or 1, given the matrix of observations X and the weight vector w.
The fit function outputs the weight vector w, given a matrix of observations X and a vector of responses y. It calls the predict function and updates the weights accordingly in a loop.

2.3 Classification - Logistic Regression

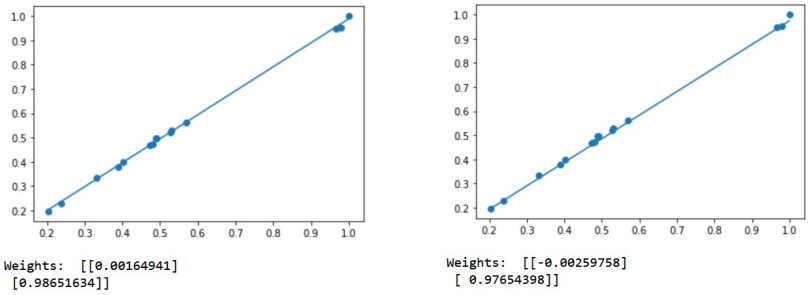
Logistic regression models the probability of an observation to belong to a class. It uses the sigmoid function implemented in the function logistic(x).
As for the perceptron, the algorithm is implemented with 2 main methods: fit and predict. The predict is based on the function predict_proba(X,w) which

returns a vector of probabilities to belong to class 1. This vector consists of $P(1|\mathbf{x}_i)$ for all the i rows of X. The threshold to belong to class 1 is 0.5.
The stop criterion is either the maximal number of epochs or the norm of the gradient.

3 Results and comments

3.1 Linear Regression

The weights and figures in Figure 2 correspond to the French dataset.
The batch descent converges faster than the stochastic descent with 236 epochs compared to 499.



(a) Batch descent (b) Stochastic descent

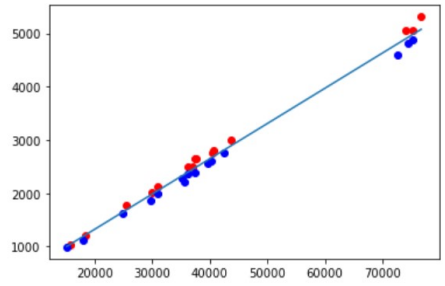
Figure 2: Regression lines obtained with linear regression

3.2 Perceptron

To evaluate the perceptron, I used the leave-one-out cross validation method. It consists of training and running 30 models. Each time, 29 samples are used for training and the remaining sample is used to validate and evaluate.
In Figure 3 one can see the weights obtained after training as well as the visualisation of the classification. Moreover, the leave one out cross validation, which consisted of training on 29 samples and evaluating on the remaining sample, got an accuracy of 96.66%.

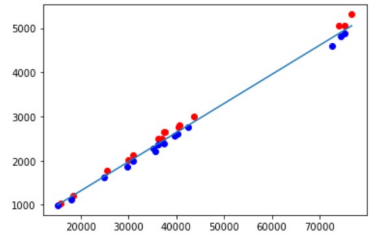
3.3 Logistic Regression

In Figure 4 a) one can see the weights obtained after training as well as the visualisation of the classification. Moreover, the leave one out cross validation, got an accuracy of 100%. Which is better than the perceptron. In b), one can see the logistic surface that the weights found before give us.
Logistic regression has a better cross validation accuracy, but the perceptron converges faster, where it stops the training before 100 epochs compared to 999 for the logistic regression.



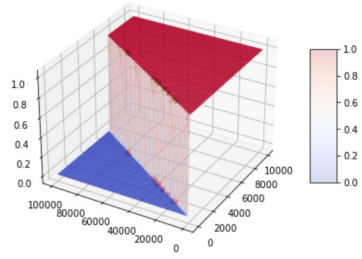

```
Weigths:  [0.0, -0.06615333883920306, 1.0]
```

Figure 3: Perceptron for the datasets




```
Weigths:  [0.0, -0.06585267290573307, 1.0]
```

(a) Logistic regression



(b) Logistic surface

Figure 4: Logistic regression for the datasets

4 Article

The article I will describe is "*An overview of gradient descent optimisation algorithms*" by Ruder (2017). I will outline the main characteristics of all the optimisation algorithms the author describes.
The classic stochastic gradient descent (SGD) does not guarantee good convergence as it can get stuck in a local minima or in a saddle point. It is also hard to choose a proper learning rate and dynamically adjust it.
In this paper, Ruder talks about several algorithms which try to solve the challenges gradient descent induces. Those algorithms are summarised below.

Momentum This method multiplies the previous step's vector with a number γ, the momentum term, when performing the update. It helps accelerate SGD in the relevant direction and dampens oscillations. This allows faster convergence and reduced oscillations, two of SGD's problems.

Nesterov accelerated gradient (NAG) This method is essentially the same as the Momentum, but with future approximation added. In the update step, it approximates the next position of the parameters and calculates the gradient with respect to those instead of the current. This method increases responsiveness.

Adagrad Adagrad dynamically updates the learning rate according to the parameters. It performs larger updates for infrequent parameters and smaller updates for frequent ones. In essence, it uses a different learning rate for every parameter. It adapts the rate at each time step based on the accumulation of the past gradients computed. It is a good method for sparse data.

Adadelta Adagrad's problem is that when updating the learning rate, it will shrink and become too small. Adadelta deals with this. This method does not accumulate all past squared gradients, but only takes a fixed number of them. It calculates a running average of past values instead of storing all of them.

RMSprop This method is also an adaptive learning rate method which is almost the same as Adadelta, but with a suggested learning rate default value.

Adaptive Moment Estimation (Adam) Adam is a combination of RMSprop and Momentum. It computes adaptive learning rates for every parameter. It stores the decaying average of past squared gradients like Adadelta and keeps the decaying average of past gradients like Momentum. Those values are estimates of the first and second moments (mean and variance).

AdaMax This is a similar method as Adam, but instead of using the second norm on the gradients for the update, it uses the infinity norm, which is considered to be stable.

Nadam Nadam combines Adam and NAG. It modifies the momentum term of Adam to be able to incorporate NAG into it. It performs a more accurate step in the descent by updating the parameters before computing the gradient.