Assignment 3

Logistic regression

Filippa Hansen

EDAP01

March 1, 2023

1 The Assignment

The assignment required the development of a linear regression program using both batch gradient descent (BGD) and stochastic gradient descent (SGD), as well as the creation of linear classifiers using the perceptron algorithm and logistic regression. The goal was to experiment with these algorithms and evaluate their performance. The provided notebook included examples that utilized Scikit Learn and Keras for predictions, which were also expected to be experimented with. Additionally, the assignment required reading and providing comments on the scientific article "An Overview of Gradient Descent Optimization Algorithms" by Ruder (2017).

2 The implementation

The first part that was to be implemented was the part that relates to linear regression. The task was to implement a program to predict the number of A's in a text from the total count of letters. This was to be done on two data sets, the English version and the French version of Salammbô. The task could be solved either by using batch gradient descent (BGD) or stochastic gradient descent (SGD). BGD updates model parameters using the average of gradients computed over the entire dataset, while SGD updates parameters using a single randomly selected data point at a time. The main difference between the two is the number of data points used to compute the gradient at each iteration.

The second part of the assignment was to classify a chapter as French or English, the dataset that was provided is the same one as for the linear regression. A pair of numbers corresponding the letter count and count of A's was given, from this the language was to be predicted. To do this the line that minimizes the sum of all squared errors is found by implementing the perceptron model. The peceptron model consists of the methods fit(X,y) and predict(X,w). The fit-method trains a model and returns the vector w, which is the weights produced given X and y. The fit-method is implemented with SGD. The stop criteria is set by the variable max_missclassified which ensures that the weights that where to be returned isn't produced in an iteration where the number of samples that are miss-classified is greater than the variable max_missclassified. The learning rate is updated for each epoch (t): alpha(t) = 1000/(1000+t). The predict method returns a vector y which contains the predicted language (in binary form). The prediction is obtained by calculating the dot product of the input arguments X and w, converting it to a 1 if the value is greater than zero or to a 0 if not.

The next step was to evaluate the peceptron using the leave-one-out cross validation method, where 30 models are to be trained and run. The leave-one-out cross validation method uses all samples except one to train the model/get the weights and the remaining sample to evaluate the prediction. This is done 30 times since there are 30 samples, the training and validation sets are updated for each session. The method uses the fit method described above (using SGD).

When the perceptron program was completed, logistic regression was to be implemented. This was done by using the SGD algorithm. The stopping criteria in this case is set to when the norm of the gradient is less then the value 'epsilon'. The prediction in this case uses the logistic method that looks like this: $y(x) = Logistic(w \cdot x) = 1/(1 + exp(-w * x))$. Where w is the weight vector and x is a sample from X.

In the last part of the notebook logistic regression classification is applied with Sckit Learn and Keras to predict the languages.

3 The example set(s) and the resulting weights

A print-out of the example set(s) and the resulting weight vectors;

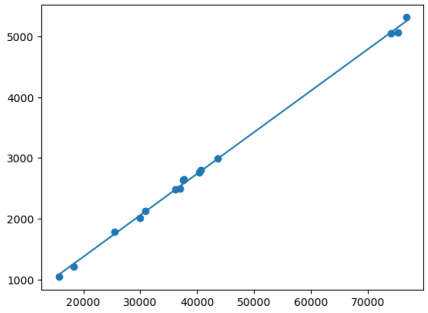

Figure 1: Batch gradient descent french, Weights [[0.00174165] [0.98635276]].

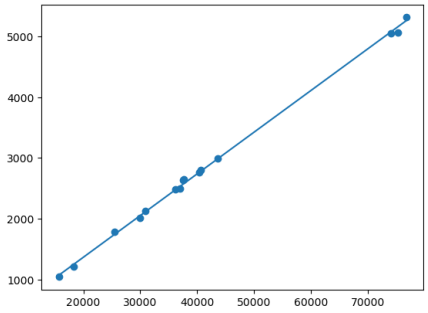

Figure 2: Stochastic gradient descent french, Weights [[-8.90269626e-04] [9.90629693e-01]].

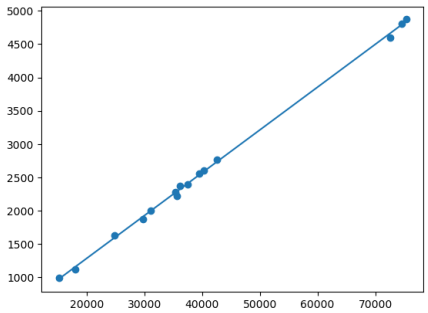

Figure 3: Batch gradient descent english, Weights [[-6.37911654e-04] [9.94530913e-01]].

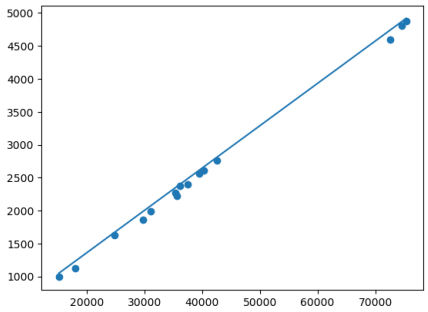

Figure 4: Stochastic gradient descent english, Weights [[0.01547424] [0.9939846]].

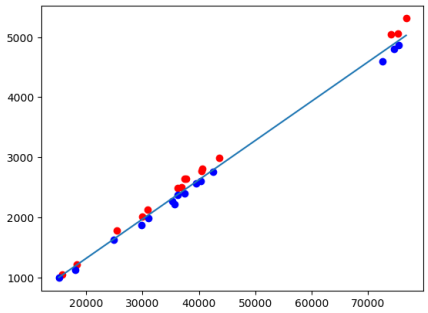

Figure 5: The results from the perceptron, Weights [[-7.896167978753164, -0.0654535073330504, 1.0]], Cross-validation accuracy (stochastic): 1.0

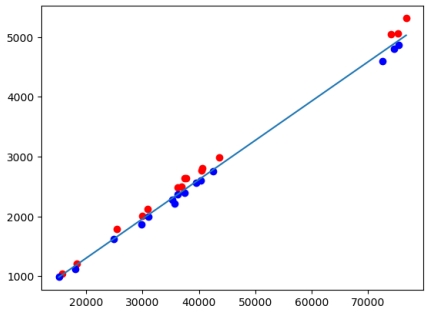

Figure 6: The results from the logistic regression, Weights [[10.21457347319949, -0.0656808178750245, 1.0]], Cross-validation accuracy (stochastic): 1.0

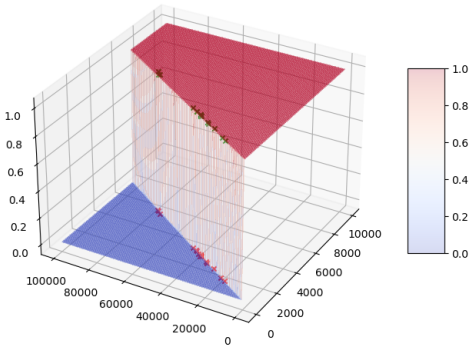

Figure 7: The logistic surface.

4 Result

The results obtained was as good as possible since a Cross-validation accuracy of 1.0 was obtained both for the Perceptron and the logistic regression.

5 An overview of gradient descent optimization algorithms

The article ”An overview of gradient descent optimization algorithms” covers a variety of optimization algorithms, including Momentum, Nesterov accelerated gradient (NAG), Adagrad, Adadelta,

RMSprop, Adam, AdaMax, and Nadam. Below the main characteristics of the optimization algorithms are described.

a) Momentum:

An extension of SGD that accumulates past gradients to accelerate learning in directions with persistent but small gradients. The algorithm adds a fraction of the previous update to the current update, which smooths out the updates and allows for faster convergence.

b) Nesterov Accelerated Gradient (NAG):

A modification of momentum that accounts for the future gradient estimates and corrects for the error that accumulates in momentum. Computes an intermediate point using the momentum update and then computes the gradient at that point to update the model weights.

c) Adagrad:

An adaptive learning rate optimization algorithm that scales the learning rate for each parameter based on the historical gradient information. Efficient for sparse data but may not work well for non-convex problems because the learning rate can get too small.

d) Adadelta:

An extension of Adagrad that addresses the problem of the learning rate getting too small by using an exponentially decaying average of the past squared gradients to estimate the second moment of the gradient. Does not require a learning rate hyperparameter.

e) RMSprop:

An adaptive learning rate optimization algorithm that uses an exponentially decaying average of the past squared gradients to scale the learning rate for each parameter. Rescales the gradient using the root mean square of past gradients.

f) Adam:

An adaptive learning rate optimization algorithm that combines the advantages of both momentum and adaptive learning rate methods. Computes an estimate of the first and second moment of the gradient to adaptively adjust the learning rate for each parameter.

g) AdaMax:

An extension of Adam that replaces the L2 norm of the gradient with the L-infinity norm. Designed to be more stable than Adam for very large models.

h) Nadam:

An extension of Adam that incorporates Nesterov momentum into the adaptive learning rate method. Computes an intermediate point using Nesterov momentum and then computes the gradient at that point to update the model weights.