Report of assignment 2

Joel Bäcker (jo4383ba-s) of EDAP01

February 23, 2022

1 The assignment

In this assignment the main goal were to perform linear regression using batch and stochastic gradient decent as well as to construct a linear classifier using a perceptron and logistic regression. The gradient decent were implemented in several different way to fit different input formats, such as a list or `numpy`-structures or to solve different problems. To estimate the performance of the models created the leave-one-out validation is used. Lastly in this report, a short dissertation of the article *An overview of gradient descent optimization algorithms* by Sebastian Ruber.

Improvements

Somethings I found hard in this assignment where

- I could not really figure out how to properly figure out the likelihood in the assignment, some more guidance would have been appreciated.
- The points for this report were quite unclear and hard to figure out what you actually want us to get out of it.

But to end on a high note, I found the slides to be an very good source to solve the problems in the assignment. It were straight forward how to go from the slides to the actual problem.

2 Implementations

The first part of the assignment relates to linear regression. The problem to be solved is to differentiate between English and French by studying the ratio between total number of characters and the number of A's chapter by chapter. There were two ways of finding the the model that could perform the prediction batch and stochastic gradient decent (SGD).

Gradient decent works by stepping in the opposite direction of the gradient. If the gradient reaches (or approaches) 0 a local minimum has been found. The difference between batch and stochastic has to do with when the update of the weights are performed. When using the first of the two the entire data set is used to update the weights, while the latter updates after each observation.

Next in the assignment, we will look at the classification problem on the same data as before. The goal of this part is to find a linear decision boundary between the two languages, which is done by finding the line that minimizes the sum of all squared errors. An error is the distance from the line to the point. To find the line a the perceptron model, described in the lecture slides, is implemented. It consists of two functions `fit(X,y)` and `predict(X, w)`.

The first function will train a model and will produce a vector of weights $\mathbf{w}$ given a training set consisting of a matrix $\mathbf{X}$ where each row contain a 1 for the models intercept, the number of letters in a chapter and the number of A's and a vector $\mathbf{y}$ that contains the correct language for corresponding row in the matrix. Finding the weight vector is found by using SGD, with a learning rate that depends on the epochs, $\alpha(t) = 1000/(1000 + t)$. The stopping criteria is set by a variable `max_misclassified`, i.e. if the model creates a vector with fewer errors than that it returns the current weight vector.

The second function, `predict(X, w)`, will produce a vector of predictions $\hat{\mathbf{y}}$ given the same vector $\mathbf{X}$ as before and a vector of weights $\mathbf{w}$. It works by calculating the multiplication $\mathbf{Xw}$, which results in a vector with value from 0 to 1. The values in the vector are then rounded to 0 or 1 and then vector is returned.

The last part of the assignment, before the section with popular APIs, we look at logistic regression to solve the same problem as before. The difference from the previous implementations is in the predict method and the stopping criteria. Instead of just doing the multiplication and then rounding, we also apply a logistic function before the rounding, the logistic function is $f(x) = 1/(1 + e^{-x})$. The other difference is the SGD is stopped when the gradient is close to zero, i.e. smaller than ϵ.

The very last part of the lab explores some popular API used for machine learning and artificial intelligence, the packages are `sklearn` and `keras`.

3 Print outs and results

See figures 1–5, for both plots and weights.

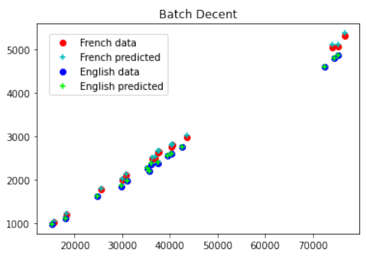

Figure 1: Batch gradient decent, weights fr: $(-0.00103, 1.01247)$ and eng: $(0.00314, 1.00103)$

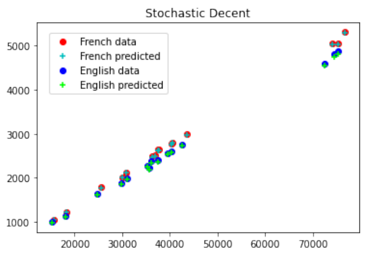

Figure 2: SGD, weights fr: (0.01007734, 0.99518834) and eng: (0.01011868, 0.98800741)

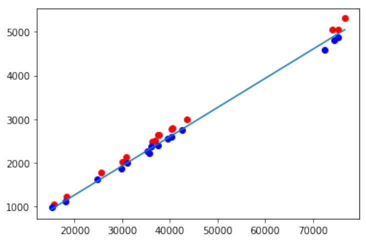

Figure 3: Perceptron, weights (75.3056532, -0.06678234, 1.0), Cross-validation accuracy (stochastic): 0.96667

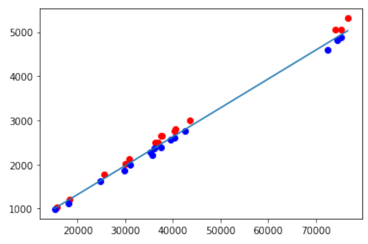

Figure 4: Perceptron logistic regression, weights (-11.551273, -0.06539033, 1.0), Cross-validation accuracy (batch): 1.0

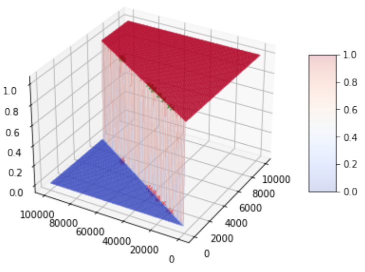

Figure 5: The logistic surface for the last part

4 Paper

In the paper *An overview of gradient descent optimization* by Sebastian Ruder different algorithms to improve the gradient decent are presented. The rest of this report aims to explain these.

The problems with SGD is the original one do not guarantee convergence, and other problems we need to consider it can be hard to chose an optimal learning rate, dynamically updating the learning rate can increase the performance and avoiding getting stuck in a suboptimal local minima. The optimizations below tries to solve this.

Momentum: adds a term to help accelerate the SGD in the right direction as well as dampens the oscillation.

Nesterov accelerated gradient (NAG): approximates the future gradient and from that decides how to update.

Adagrad: dynamically adapts the learning rate to perform larger updates for infrequent parameters and smaller updates on more frequent parameters.

Adadelta: is an extension of the previous optimization, where this method restricts the summation of all the previous gradients by instead looking at a fixed window of size w. This is achieved not by storing all w values but instead as a recursively defined decaying average of all past square gradients.

RMSprop: is very similar to Adadelta but with a suggested learning rate of 0.9.

Adam: combines the decaying average squared gradient, like Adadelta, and decaying average gradients, like momentum. Similar to Adagrad, parameter's learning rates are adaptivly found.

AdaMax: Similar to the regular Adam-optimizer, but instead of using the second norm on previous and current gradients, the infinity one is used. The infinity norm, as well as the first and second are considered numerically stable, hence why they can be used.

Nadam: modifies Adam-optimizer in order to include the NAG, i.e. it is a combination of the two.

The paper also mentions parallelizations of the SGD and names a few optimizers such as Hogwild, Downpour SGD and TensorFlow. By distributing the computation the speed of which it find a minima increases.