EDAP01 Assignment 2 - Linear regression

Olivia Mattsson (ol3270ma-s)

February 2021

1 Introduction

The assignment gave an introduction to linear and logical regression, gradient descends, and how they are implemented in Artificial Intelligence. It presented ways of classifying data according to patterns that has been observed in training materials, or data-sets. Besides implementing these algorithms ourselves, the assignment introduced additional built in tools in Python used to solve these issues.

2 Implementation

Besides the implementation for both gradient descents, the stochastic version was the chosen one throughout the assignments. When the initial gradient descent implementations were tested, the regression lines from the stochastic and batch gradient descents differed. From the first initial tests on the datasets, the stochastic linear regression seemed to give more viable results. Given that it also scales better with big datasets, it was the chosen version.

When comparing the result to the given solutions for the linear regressions, they differed a bit. But when checking the `sse()` function, the error rate overlapped with the given result. After a discussion with one of the TA's, it was deemed plausible for the datasets and the differences would lie in the implementations. Both the English, Figure 1, and the French, Figure 2, gave plausible results. The regression lines followed the datapoints in a good way and produced a model for both batch and stochastic gradient descend. If observed carefully, the stochastic regression line seemed to be more accurate than the batch version, going through more points. Both of them used the given values of alpha, 1, and used the normalized datasets before turning it back to the given values.

The Perceptron was used for classification of the different chapters to decide if they was written in English or French. The implemented predict function, given the weight and the input, performs a matrix multiplication and returns 1 if it predicts the chapter was written in French, and a 0 for English. If the matrix multiplication were a negative or equal to 0, then English was returned

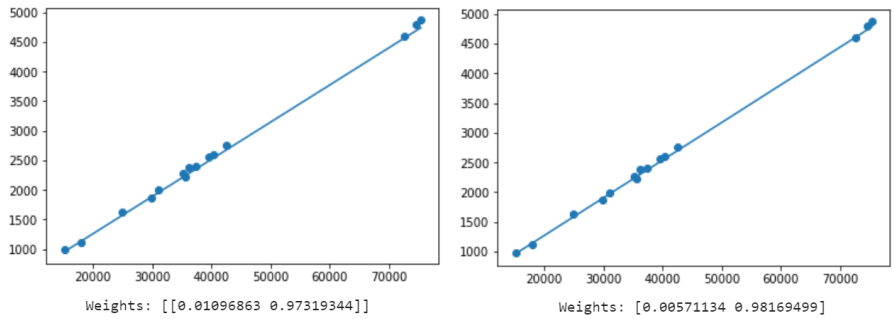


Figure 1: From the right: Batch and stochastic linear regression for the English dataset with their weights.

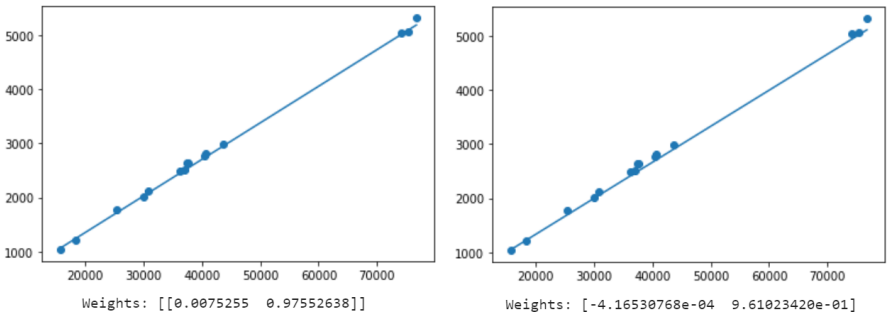
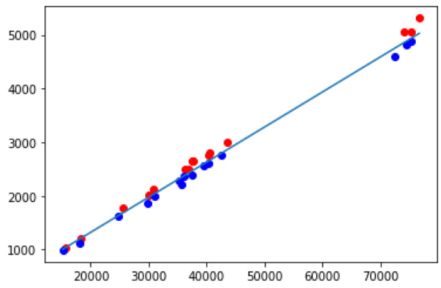


Figure 2: From the right: Batch and stochastic linear regression for the French dataset with their weights.

as the prediction, and French otherwise.
The Perception function loops over the dataset using the stochastic gradient descend, but instead uses the newly introduced prediction function to predict the language of each chapter. If it produces more misclassifications than one, then it will continue to loop until the condition is met or the `max_iter` has been reached. The new weight is produced with the difference between the real value and our prediction, to finally stabilize at a reasonable value. As seen in Figure 3, the model for the given input produced a good model of the classification. The evaluation concluded in no missing predictions.
The learning rate was used to minimize the gradient's influence for each iteration, and also was taken into consideration when choosing the new w. The learn rate was multiplied with the gradient to make sure that it didn't compensate to much.

The assignment continued with a logistic regression. Instead of using the predic-




```
Cross-validation accuracy (stochastic): (30, 30)

  [ -0.0838294 , -26.00871201,  27.58045141]
```

Figure 3: Perceptron for the French and English datasets

tion function from the Preceptron, the function `logistic()` was used to predict the value. The difference between the real value and the prediction is then again added to the weight, resulting in a model that is plotted between the two different types of datapoints in Figure 4. When implementing a stop criteria, the model did not look similar to this, and is not kept in this implementation. Both of the variants where tested, and produced faulty regression lines. If given more time, I would continue to find the correct way of interrupting the loop if a good enough w was produced.
The evaluation showed that the accuracy, once again, was 100%. The evaluation this time used a prediction function with the calculated weight to see if it would predict the given point correctly, using the functionc `predict()` and `predict_proba()`. The current implementation only uses one datapoint when predicting, and is something that could be done differently.

Finally, the different built in packages were examined and tested. They ran smoothly in the Notebook browser after the packages were installed. Both `sklearn` and `Keras` worked good and predicted each chapter perfectly. However, in Keras the epochs were set to 2 000 instead of 1 500 to make sure that the correct predictions happened. With the initial 1 500 epochs, there were some misclassifications.

3 Article

The article as a whole covers the concept of how gradient descents are implemented in Deep Learning libraries. It gives some information about the different types of gradient descent, difficulties with training, and presents a variety of optimization algorithms using them.

Section 2 presents three different types of gradient descent - batch, stochastic, and mini-batch gradient descent. The first two has been implemented in this report, while the latter has not yet been covered.

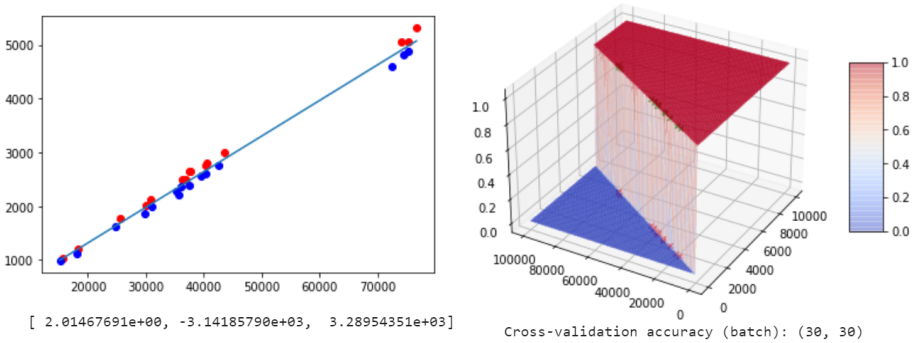


Figure 4: Logistic regression for the French and English datasets

In general, a gradient descent takes the input from a dataset and tries to find the local minimum for some function by moving the opposite direction of the gradient. In linear regression, it is used to update the coefficients. It will have some learning rate that defines how much you move in each iteration towards the minimum. This can affect both the speed of the function but also whether or not it can converge.

The batch gradient descent takes the entire dataset that is given for training and computes the gradient of the function. Since the entire dataset is used for a single update, it is computationally heavy and is not to be preferred on large datasets.

The stochastic gradient descent, in contrast to the batch version, computes one parameter update at a time instead of looping over the entire dataset. This enables stochastic to find other better local minimas, while batch will only find the minimum of the current basin where the parameters has been placed in.
Mini-batch gradient descent combines both of them. The dataset is divieded into smaller sets, or mini-batches, and an update if performed on each of them. The dataset is shuffled in the same manner as in SGD, but then the gradient is calculated only on one batch at a time. This gives Mini-batch gradient descent some perks and is the most common algorithm in training of a neural network.

Even though mini-batch gradient descent is the preferable choice according to this paper, chapter three presents some important issues to consider when implementing the alorithm. Choosing a good learning rate is presented as an important part. If the rate is chosen as too large, it might diverge instead of finding the local minimas. One the other hand, if the rate is too small the convergence will take too long. The use of learning rate schedules can be nice to adapt the learning rate depending on how much the parameters change between each epoch. But they can not, however, be adapted to the characteristics of the dataset. All parameter updates will use the same learning rate, which might be problematic when the dataset is small.

Other than the learning rate, the chapter also talks about the possibility for the algorithms to get stuck in a saddle points that will be interpreted as a minima to the algorithm. The error rate will be the same on all sides and the gradient close to zero, which can be interpreted as a minima.

The report presents several popular algorithms that utilises the gradient descent algorithms, and addresses the challenges presented in chapter three.
The first algorithm that is presented in chapter four is Momentum. It presents a variable γ in the update vector that helps guide the SGD towards the right direction. Like the name suggests, it uses the metaphore of a ball falling down a slope. The γ term is the velocity that increases, called the momentum term. Whenever two gradients point in the same direction, we will increase the momentum in that direction, leading to a quicker convergence towrads the desired value.

But an important aspect that is later brought up is that the metaphorical ball would ideally know the direction and knows to slow down and not follow the slope in upward directions. The Nesterov accelerated gradient (NAG) is then presented, which solves this issue and gives this awareness to the momentum term. NAG calculates the current parameters, and also predicts the next parameters. NAG then uses this approximation to prevent compensating for large steps in the wrong direction and speeds up the SGD.

The next algorithm that is presented addresses the challenge of updating the parameters differently. Adagrad performs bigger updates on the less frequent parameters, and lesser on the more frequent ones. This implementation works well for dealing with small datasets. The perks of Adagrad is that the learning rate does not need to be modified manually. But as the dataset used for training is processed, the gradients will continue to grow and causing the new gradients to barely change, and it stops its learning process. Adadelta is an extenstion that aims to solve this problem, where it limits the sum of all of the accumulated gradients in set window sizes w. It managed to remove the decreasing learning rate by introducing the windows of accumulated gradients.

The chapter later on briefly mentions the method RMSprop, an unpublished method that was developed around the same time as Adagrad. RMSprop have similarities with Adagrad, and also utilizes the sum of accumulated gradients that modifies the learning rate accordingly. Adam, an acronym for Adaptive Moment Estimation, could be seen as a combination of momentum and RMSprop where both the past squared and past gradients are used. It is shown to perform well in practice. AdaMax, an extension to Adam, presents more stable values than Adam's. Nadam is another extension that combines NAG and Adam.

The chapter ends with a visualization of the behavior of the algorithms, and a comparison between them. One of the conclusions is that the ordinary SGD is often preferred and implemented without momentum and a simple learning rate. It might take longer, but it usually finds the minimum. But if a fast

convergence is preferred then one of the optimizations should be used, where Adam is the best overall choice.