Artificial Intelligence Assignment 2

Classification with the Perceptron and Logistic Regression

Name: Isabelle Frodé
Date: March 10 2021

1 Assignment Overview

The first task of this assignment was to write a program for *linear regression* using gradient descent. Regression lines were computed using batch gradient descent and stochastic gradient descent. The next part of the assignment was to implement *linear classifiers* using the perceptron algorithm and logistic regression. The batch or the stochastic version of the algorithm could be used, student's choice. The final part of the assignment was to apply logistic regression using two commonly used tools: sklearn and keras.

Possible *improvements of the assignment* that I would suggest is mainly some clarification. For example, the sentence in the report description: "...the possible improvements you would have brought" could refer to improvements of the actual assignment but also improvements of *my own work* within the assignment. It was also not clear how detailed the report should be. A page limit would have made it easier to understand the expectations on the report.

Possible *improvements of my work within the assignment* if I had more time would be to improve my implementation. I am more used to programming in for example Matlab. The code could definitely get more neat with more practice in Python and it is certainly possible to reduce the number of lines used.

2 Implementation

2.1 Linear Regression

To predict the counts of the letter A in a text, a regression line was computed using gradient descents methods. The algorithms were described in *Artificial Intelligence: A Modern Approach* by Russell-Norvig [1].

The feature matrix $\mathbf{X}$ and the $\mathbf{y}$ vector was extracted from the dataset given in the assignment [2] and then scaled to fit in the interval $I = [0, 1]$. The feature matrix $\mathbf{X}$ is the input and the $\mathbf{y}$ vector is the output.

The gradient descent functions takes as input the $\mathbf{X}$ matrix, the $\mathbf{y}$ vector, the learning rate α, the initial weight function $\mathbf{w}$, the tolerance ϵ and the maximal number of epochs *epochs*. The function then outputs the model containing the weights $\mathbf{w}$ that minimizes the partial derivative of the loss.

The stop condition for the descent methods was defined by calculating the gradient of the loss.

2.1.1 Batch Descent

The batch descent algorithm compute the gradient of the whole dataset for each update [1]. The weights **w** are updated by using equation 1 and 2 for the entire dataset for every update.

$$w_0 \leftarrow w_0 + \frac{\alpha}{q} \sum (y^j - (w_0 + w_1 x_1^j)) \tag{1}$$

$$w_1 \leftarrow w_1 + \frac{\alpha}{q} \sum x_1 \times (y^j - (w_0 + w_1 x_1^j)) \tag{2}$$

2.1.2 Stochastic Descent

The stochastic descent algorithm update for one observation at the time and is suitable for online computing [1]. The weights **w** were updated using equation 3 and 4.

$$w_0 \leftarrow w_0 + \alpha \cdot (y^j - (w_0 + w_1 x_1^j)) \tag{3}$$

$$w_1 \leftarrow w_1 + \alpha \cdot x_1^j \cdot (y^j - (w_0 + w_1 x_1^j)) \tag{4}$$

2.2 Linear Classifiers

Two different linear classifiers, which decides which out of two categories a data point belongs to, were implemented. The algorithms was evaluated using an implementation of leave-one-out cross validation using 30 models. 29 samples were trained and a 30:th sample was used for evaluation.

2.2.1 Perceptron Algorithm

The perceptron learning rule was implemented as described in Russel-Norvig [1]. The function takes as input the **X** matrix and the weights **w** and returns a prediction of the class property. A fit function that returns a model of the weights **w** which fit the data was implemented using stochastic descent. The weight were updated by using equation 5.

$$w_i \leftarrow w_i + \alpha \sum x_i^j (y^j - \hat{y}^j) \tag{5}$$

The stop condition was defined by a predefined max value of misclassified examples, which was set to 0.

The classification was done as described in equation 6.

$$w_0 + w_1 x_1 + w_2 x_2 = \begin{cases} < 0 \implies & \text{belong to class 1} \\ > 0 \implies & \text{belong to class 2} \end{cases} \tag{6}$$

2.2.2 Logistic Regression

Logistic regression was then implemented to classify the data points using batch descent. The logistic regression takes a vector of probabilities of data points to belong to a class. The probability is determined by the logistic function shown in equation 7.

$$P(y = 1|x) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}} \tag{7}$$

A stochastic descent method was then implemented as the fitting function to fit the model to the data. The weights was updated using equation 8.

$$w_i \leftarrow w_i + \alpha \cdot x_1^j \cdot (y^j - (w_0 + w_1 x_1^j)) \tag{8}$$

3 Example Datasets

The example datasets were given in the assignment description [2]. The dataset used for the linear regression is shown in Figure 1 and the dataset used for the linear classifiers is shown in Figure 2.

```
stat_fr = np.array([[36961, 2503],
                    [43621, 2992],
                    [15694, 1042],
                    [36231, 2487],
                    [29945, 2014],
                    [40588, 2805],
                    [75255, 5062],
                    [37709, 2643],
                    [30899, 2126],
                    [25486, 1784],
                    [37497, 2641],
                    [40398, 2766],
                    [74105, 5047],
                    [76725, 5312],
                    [18317, 1215]])
```

(a) French

```
stat_en = np.array([[35680, 2217],
                    [42514, 2761],
                    [15162, 990],
                    [35298, 2274],
                    [29800, 1865],
                    [40255, 2606],
                    [74532, 4805],
                    [37464, 2396],
                    [31030, 1993],
                    [24843, 1627],
                    [36172, 2375],
                    [39552, 2560],
                    [72545, 4597],
                    [75352, 4871],
                    [18031, 1119]])
```

(b) English

Figure 1: Example datasets containing letter counts in the 15 chapter of Salammbô. Left column show total character count and right column show count of 'A'

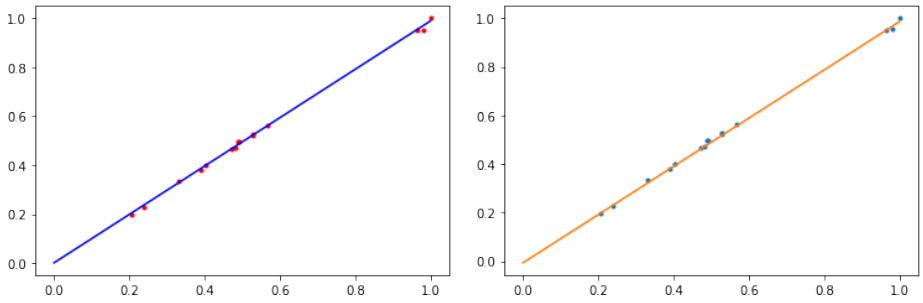
```
X = [[1.0, 35680.0, 2217.0],
     [1.0, 42514.0, 2761.0],
     [1.0, 15162.0, 990.0],
     [1.0, 35298.0, 2274.0],
     [1.0, 29800.0, 1865.0],
     [1.0, 40255.0, 2606.0],
     [1.0, 74532.0, 4805.0],
     [1.0, 37464.0, 2396.0],
     [1.0, 31030.0, 1993.0],
     [1.0, 24843.0, 1627.0],
     [1.0, 36172.0, 2375.0],
     [1.0, 39552.0, 2560.0],
     [1.0, 72545.0, 4597.0],
     [1.0, 75352.0, 4871.0],
     [1.0, 18031.0, 1119.0],
     [1.0, 36961.0, 2503.0],
     [1.0, 43621.0, 2992.0],
     [1.0, 15694.0, 1042.0],
     [1.0, 36231.0, 2487.0],
     [1.0, 29945.0, 2014.0],
     [1.0, 40588.0, 2805.0],
     [1.0, 75255.0, 5062.0],
     [1.0, 37709.0, 2643.0],
     [1.0, 30899.0, 2126.0],
     [1.0, 25486.0, 1784.0],
     [1.0, 37497.0, 2641.0],
     [1.0, 40398.0, 2766.0],
     [1.0, 74105.0, 5047.0],
     [1.0, 76725.0, 5312.0],
     [1.0, 18317.0, 1215.0]]
y = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
     1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

Figure 2: Example dataset where **X** contains the count of characters and count of letter A. **y** contains the classes, where 0 is for English and 1 is for French

4 Results

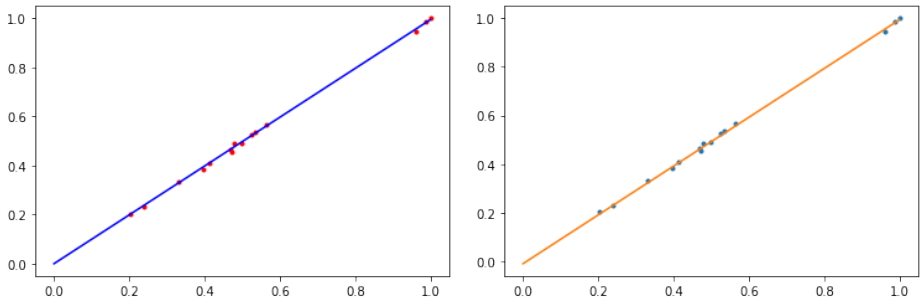
4.1 Linear Regression

The visualization of the resulting weight vectors after applying gradient descent are shown in Figure 3 and Figure 4. The Figures show the case when the learning rate $\alpha = 1$ and the initial weights $\mathbf{w} = [1, 1]$.



(a) Batch descent **(b)** Stochastic descent

Figure 3: French dataset with the obtained weight vector as the regression line

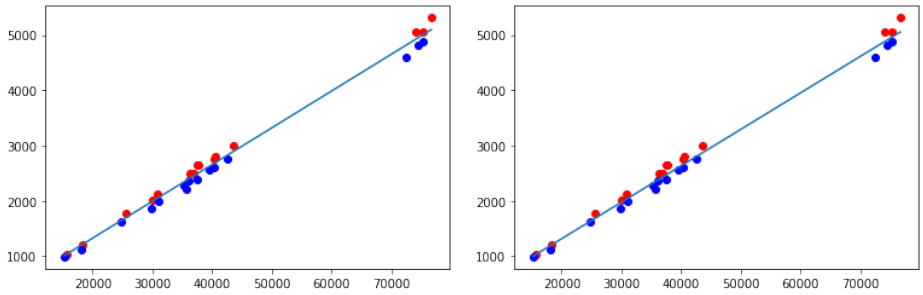


(a) Batch descent **(b)** Stochastic descent

Figure 4: English dataset with the obtained weight vector as the regression line

4.2 Linear Classifiers

The visulaization of the resulting weight vectors after applying the perceptron algorithm and the logistic regression are shown in Figure 5.



(a) The Perceptron Algorithm **(b)** Logistic Regression

Figure 5: Classification of French (blue), English (red) and the obtained weight vector

The logistic surface after applying logistic regression is visualized in a 3D plot shown in Figure 6.

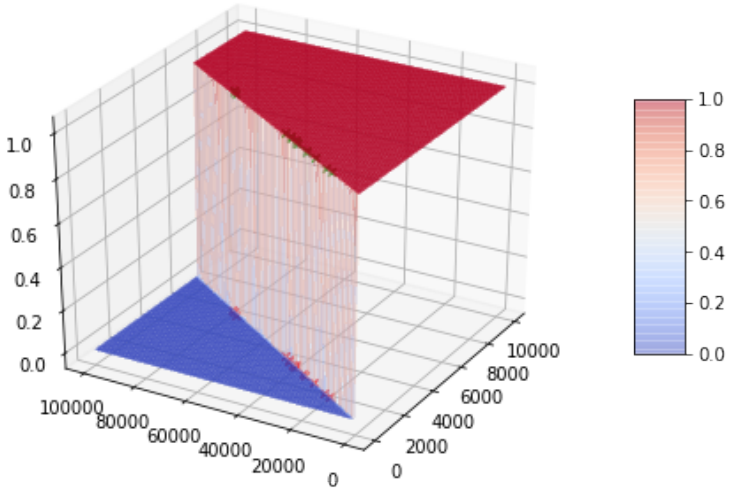


Figure 6: Visualization of the logistic surface

The cross-validation accuracy with the leave-out-out method for the perceptron is shown in Figure 7 and for the logistic regression in Figure 8.

```
Fold  0 weights:  [  1.         -80.50245683  82.00451807] :,  correct
Fold  1 weights:  [   0.         -139.05393288  146.09167922] :,  correct
Fold  2 weights:  [ -1.         -48.24439231  52.42074548] :,  correct
Fold  3 weights:  [  1.         -70.23816227  70.83716114] :,  correct
Fold  4 weights:  [   0.         -125.18147931  131.99435241] :,  correct
Fold  5 weights:  [ -1.         -53.07857934  57.3721762 ] :,  correct
Fold  6 weights:  [  -5.         -152.73329423  167.80064006] :,  correct
Fold  7 weights:  [  0.         -21.34475073  22.1842997 ] :,  correct
Fold  8 weights:  [   0.         -121.40130336  127.26976657] :,  correct
Fold  9 weights:  [  0.         -36.05676116  37.67206325] :,  correct
Fold  10 weights:  [ -1.         -99.86266536 105.96253765] :,  correct
Fold  11 weights:  [  -3.         -120.80655588  129.77315512] :,  correct
Fold  12 weights:  [  -3.         -154.46026719  168.02108434] :,  correct
Fold  13 weights:  [  -5.         -152.12993157  166.44164157] :,  correct
Fold  14 weights:  [  0.         -54.89850766  57.87612952] :,  correct
Fold  15 weights:  [   0.         -130.07762789  137.11182229] :,  correct
Fold  16 weights:  [  0.         -36.89243402  38.74792922] :,  correct
Fold  17 weights:  [ -1.         -91.8092799   98.53012048] :,  wrong
Fold  18 weights:  [   0.         -122.16514826  128.5188253 ] :,  correct
Fold  19 weights:  [  -2.         -139.27158032  149.91246235] :,  correct
Fold  20 weights:  [ -2.         -83.71204953  89.59789157] :,  correct
Fold  21 weights:  [  0.         -73.15378299  76.78915663] :,  correct
Fold  22 weights:  [ -1.         -80.88059954  87.45500753] :,  correct
Fold  23 weights:  [   0.         -131.97373737  137.90530873] :,  correct
Fold  24 weights:  [  -2.         -106.43327468  116.18335843] :,  correct
Fold  25 weights:  [   1.         -102.76639948  106.3217244 ] :,  correct
Fold  26 weights:  [  -1.         -130.0226002   137.31626506] :,  correct
Fold  27 weights:  [  2.         -35.4283871   32.29706325] :,  wrong
Fold  28 weights:  [  1.         -68.04035191  70.06908886] :,  correct
Fold  29 weights:  [  0.         -83.66544151  87.39231928] :,  correct
Perceptron
Cross-validation accuracy (stochastic): 0.9333333333333333
```

Figure 7: Leave-one-out cross validation accuracy using the perceptron

```
Fold  0 weights:  [    726 -493099 7447400] :,  correct
Fold  1 weights:  [   -589 -394265 5958458] :,  correct
Fold  2 weights:  [    433 -364873 5555824] :,  correct
Fold  3 weights:  [   -943 -323074 4879968] :,  correct
Fold  4 weights:  [   -1492 -1435031 21690408] :,  correct
Fold  5 weights:  [   -3340 -2157015 32792104] :,  correct
Fold  6 weights:  [  -13125 -1369199 20786358] :,  correct
Fold  7 weights:  [    202 -385581 5832958] :,  correct
Fold  8 weights:  [    -637 -1349494 20366533] :,  correct
Fold  9 weights:  [     51 -608803 9199542] :,  correct
Fold  10 weights:  [     -52  -787352 11905598] :,  correct
Fold  11 weights:  [    -950  -706537 10715815] :,  correct
Fold  12 weights:  [  -3502 -524647 7916478] :,  correct
Fold  13 weights:  [   -977 -399495 6079120] :,  correct
Fold  14 weights:  [   1493 -617370 9319264] :,  correct
Fold  15 weights:  [   -5055 -1896330 28724556] :,  correct
Fold  16 weights:  [   -4287  -760165 11532362] :,  correct
Fold  17 weights:  [   -6576 -1603818 24348570] :,  correct
Fold  18 weights:  [   -2712  -828746 12560750] :,  correct
Fold  19 weights:  [   -3810  -786465 11933594] :,  correct
Fold  20 weights:  [  -1442 -538371 8160729] :,  correct
Fold  21 weights:  [   2367 -405389 6146192] :,  correct
Fold  22 weights:  [   -536 -483963 7332888] :,  correct
Fold  23 weights:  [   -722 -639715 9708865] :,  correct
Fold  24 weights:  [   -4203 -1523125 23112629] :,  correct
Fold  25 weights:  [    -142  -798466 12040387] :,  correct
Fold  26 weights:  [    -252 -1020490 15384879] :,  correct
Fold  27 weights:  [    7683 -1380107 20854932] :,  correct
Fold  28 weights:  [    6617 -1517367 22911172] :,  correct
Fold  29 weights:  [   -2866 -1171596 17774700] :,  correct
Logistic Regression
Cross-validation accuracy (stochastic):  1.0
```

Figure 8: Leave-one-out cross validation accuracy using the logistic regression

5 Comments on Results

The results show that the implemented code seem to be doing its job. The linear regression lines obtained by both batch and stochastic descent shown in Figure 3 and Figure 4 match the data points well.

Both the perceptron and the logistic regression classification algorithms manage to classify the data very well as shown in Figure 5. The cross-validation show that the logistic regression performs even better than the perceptron, having a more accurate result which was also expected. For some runs, as shown in Figure 8, the logistic regression performs perfectly, having an accuracy of 1.0.

6 Paper Dissertation

The paper *An overview of gradient descent optimization algorithms* [3] by S Ruder describes three gradient descent algorithms: batch descent, stochastic descent and mini-batch gradient descent. The paper further aim to explain how these gradient descent algorithms can be optimized using different methods.

6.1 Gradient Descent Methods

Batch gradient descent is one of the most classical and popular methods, computing the gradients for the entire dataset for each update. The method is very slow and only allow for offline computing, however it guarantees to converge to global optima for convex error and non-convex surfaces.

Stochastic gradient descent performs a parameter update for each step and it is therefor much more efficient and fast. A downside with the stochastic gradient descent is that it does not always converge to an exact optima.

Mini-batch gradient descent combines batch and stochastic gradient descent by computing the gradient for n training examples for each update. The mini-batch descent is a good compromise between speed and accuracy.

6.2 Gradient Descent Optimization Algorithms

Momentum is an algorithm to reduce oscillations that appears when a stochastic gradient descent encounter an area with a ravine. These areas usually appear close to a local optima. The momentum algorithm helps to accelerate the stochastic gradient descent in the right direction.

Nesterov accelerated gradient is an algorithm that increases the responsiveness of stochastic gradient descent by predicting the gradient. The algorithm make stochastic gradient descent go in the previous calculated direction and then it measure the actual gradient adjust it.

Adagrad is an algorithm good for sparse data which adjust the learning rate based on the importance. For frequent parameters the algorithm make stochastic gradient descentto perform smaller updates and for less frequent parameters larger updates. By doing this, adagram take away the need of manually adjusting the learning rate.

Adadelta extends the Adagrad algorithm by preventing the learning rate to decrease to much. This is done by the Adadelta algorithm that restricts the number of accumulated past squared gradients.

RMSprop is an alternative to Adadelta. It is an algorithm that prevents the learning rate to decrease to much when using Adagrad. This is done by the RMSprop algorithm that calculate the average of squared past gradients.

Adam is another adaptive learning rate optimization method. Adam is combining RMSprop and Momentum and it could be seen as an extension of Adadelta and RMSprop by adding a exponentially decaying average of past gradients.

AdaMax is an algorithm generalizing Adam from the L_2 norm of the past gradients to the L_p norm. Common practice is to use the L_1 or L_2 norm since large values of p can lead to unstable behavior. However if we use the L_∞ norm a more stable value can be obtained.

Nadam is an algorithm that incorporates the Nestrov accelerated gradient into Adam by reforming the momentum decay term to update the current parameters.

References

[1] P Norvig, S Russel. *Artificial Intelligence: A Modern Approach. 3rd ed.* 2010. Upper Saddle River, NJ: Prentice Hall

[2] P Nugues. *Assignment Instructions: Classification with the perceptron and logistic regression.* LTH. Downloaded: 2021-02-10. https://github.com/pnugues/edap01/tree/master/assignments

[3] S Ruder. *An overview of gradient descent optimization algorithms.* 2017. arXiv:1609.04747v2 [cs.LG] 15 Jun 2017