Assignment 3: Logistic regression

Måns Alklint

March 13, 2024

1 Introduction

The assignment involved implementing linear regression using gradient descent, as well as the perceptron algorithm and logistic regression for the task of analyzing 15 chapters of the book Salammbô, to predict the counts of A's in the text based on the total count of letters. We applied these methods to two datasets corresponding to the French and English versions of Salammbô, where the first column represented the total count of characters and the second column represented the count of A's. The goal was to develop models that could accurately classify a chapter as either French or English. Given a pair of numbers corresponding to the letter count and count of A's, the task was to predict the language. This is an example how this was made:

1. $\mathbf{x} = (35680, 2217) \rightarrow y = \text{English}$
2. $\mathbf{x} = (37497, 2641) \rightarrow y = \text{French}$

To achieve this, the rough approach was taken using the following steps:

1. Utilize the gradient descent method to implement linear regression.
2. Implement the perceptron algorithm.
3. Improve the perceptron algorithm using logistical regression.

Possible improvements and as an interesting comparison, would be to use a multi-layered neural network instead of the legacy perceptron. Deep Neural Networks would offer a better linguistic analysis than that of the perceptron (being a linear model, cannot effectively learn). Deep Neural Networks can model non-linear relationships between the input features in this case character count and 'A' count, and the output language, through the use of non-linear activation functions in its hidden layers. The perceptron however, relies on a linear decision boundary, limiting its capabilities for this non-linear language prediction task. Also, discussed in the analysis there are room for far more algorithm that boasts performance, speed and reliabilty even for larger scale applications. See 5. Analysis.

2 Implementation

Throughout this process, the influence on the learning speed and accuracy was a major focus point, as various configurations were experimented with. For instance, in the gradient descent implementations, different techniques such as the batch and stochastic variants were used so that their efficiency and generalization performance could be observed.

In the case of logistic regression, I analyzed the impact of various stopping criteria on the results. I set a maximum number of misclassified examples as the stopping point and compared this with the convergence criteria, specifically the norm of the difference between consecutive weight vectors. This comparison allowed me to see the influence of both methods.

Throughout these experiments, evaluations were performed using leave-one-out cross-validation to determine the model's generalization capabilities accurately. The goal was somewhat to find optimal configurations that both is a balance between training speed and predictive performance. Since I had the book "AI: A Modern Approach" by Russell and Norvig, provided steps helped guide the design choices.

3 Examples

This example set is the combined set of french and english:

```
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])
```

This x-set represents a dataset with 30 observations, each containing three values. The first value is always 1.0, representing the intercept term in a linear model. The second value is the count of characters, and the third represent counts of the letter "A".

```
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]
```

This y-set refer to the categories or labels assigned to each observation. In this specific dataset, represented by the vector y, the classes are either 0 or 1. Each observation is assigned either English (class 0) or French (class 1).

After normalizing and fitting the dataset through a perceptron model, we obtain the following output:

```
Restored weights [-51.993242650025635, -0.43131337948424925, 6.585299300200241]
Weights with y set to 1 [-7.8953499726951915, -0.06549639732716996, 1.0]
```

This output shows the weight vectors as produced by the perceptron model. The numbers in the "Restored weights" represent the importance of each feature in determining whether a text is in English or French. The first number, approximately -52, indicates the significance of a baseline factor, like an initial assumption. The second number, around -0.43, suggests how much the count of total characters affects the decision. The third number, approximately 6.59, shows the impact of the count of the letter "A" on the classification.

In the "Weights with y set to 1", the numbers are normalized to emphasize the impact of the count of the letter "A" when determining if the text is in English. The first number, about -7.9, still represents the baseline factor, but its magnitude is adjusted relative to the others. The second number, close to -0.07, indicates the influence of total characters on the classification. Finally, the third number, 1.0, serves as a reference point for comparison, showing the importance of the count of "A" when the text is English.

4 Results

The cross-validation results from the perceptron model itself yielded a score of 29 out of 30 correct classifications ($\tilde{9}7\%$ accurate), indicating a high level of accuracy and robustness in the model's predictions. With the perceptron modified to do logistic regression, it was yet more robust with a score of 30 out of 30 correct classifications ($\tilde{1}00\%$ accurate).

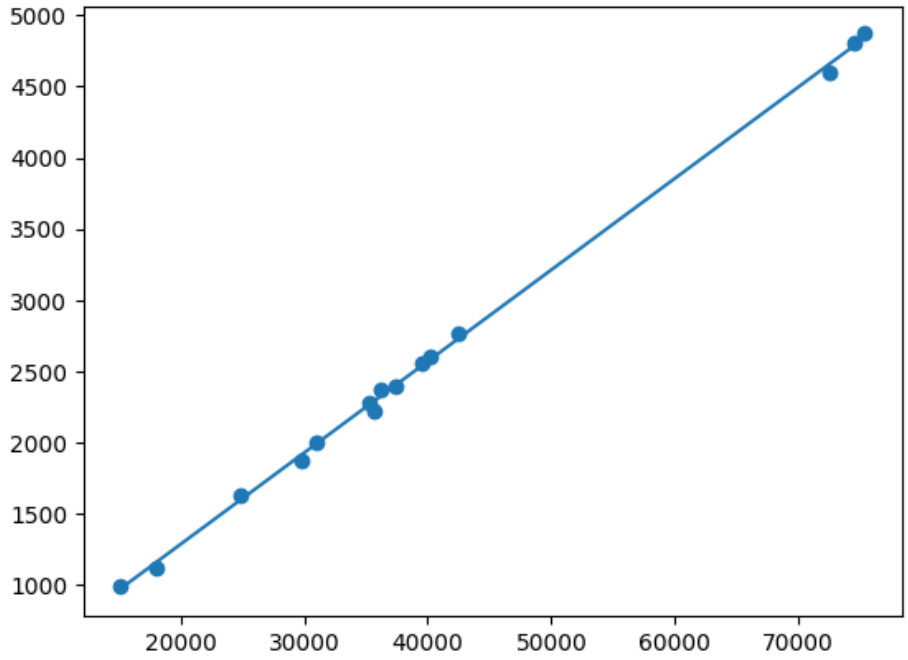

Figure 1: The linear regression of the dataset

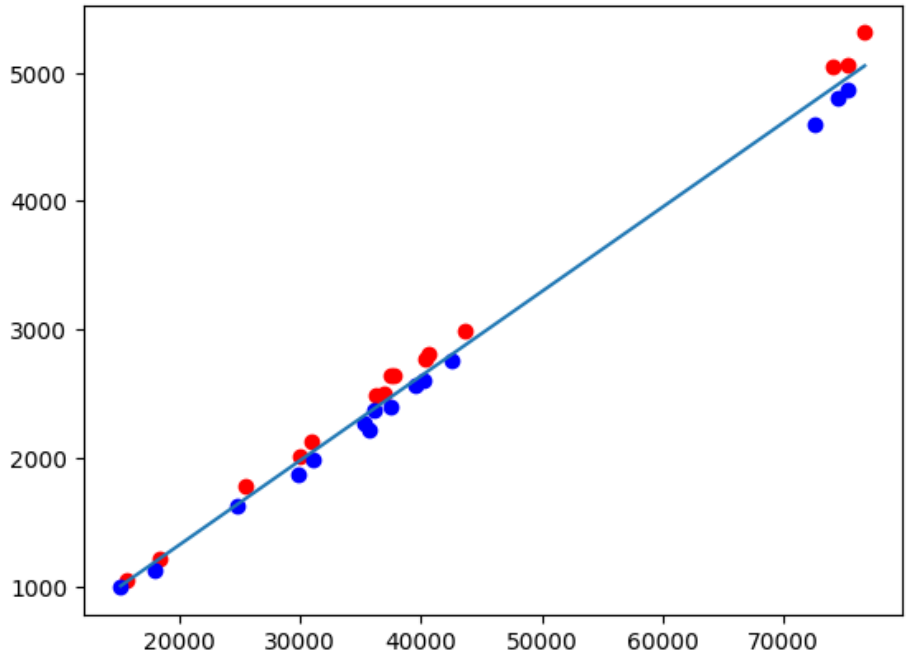

Figure 2: Visualization of the decision boundary of logistical regression based on the perceptron algorithm

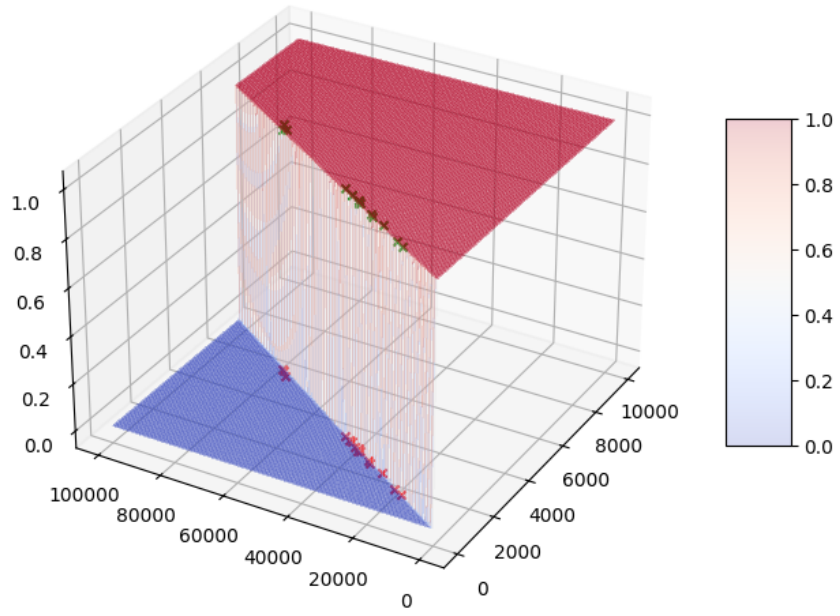

Figure 3: The probability surface as a function of x and y

5 Analysis

Ruder's paper provides a comprehensive overview of various gradient descent optimization algorithms used for training deep neural networks. This dissertation aims to provide insights into the behavior, uses and benefits of different optimization algorithms.

5.1 Vanilla SGD

The fundamental algorithm is stochastic gradient descent, SGD, which performs parameter updates for each training example to reach the minimum of the loss function. However, vanilla SGD has some challenges like choosing an appropriate fixed learning rate and getting trapped in saddle points of the non-convex loss surface.

5.2 Momentum

Momentum adds a fraction of the previous update vector to the current update to accelerate SGD in the relevant direction and dampen oscillations.

5.3 NAG

Or, Nesterov Accelerated gradient, gives the momentum term a sense of future direction by computing the gradient not at the current parameters but at the approximate future position.

5.4 Adagrad

Adagrad adapts the learning rate for each parameter based on the historical sum of their squared gradients, performing larger updates for infrequent parameters and smaller updates for frequent ones. However, its monotonically decreasing learning rate is undesirable. This alghorithm was used at Google and supposedly learned to recognize cats in Youtube videos.

Adadelta and RMSprop are extensions of Adagrad that aim to reduce its aggressive, monotonically decreasing learning rate by restricting the window of accumulated squared gradients.

5.5 Adam

Adam computes adaptive learning rates for each parameter by keeping an exponentially decaying average of past gradients (like momentum) and past squared gradients (like Adagrad/RMSrop). It combines the strengths of momentum and RMSprop.

5.6 AdaMax

AdaMax is a variant of Adam that uses the infinity norm and is slightly more robust to non-stationary objectives. Nadam incorporates Nesterov momentum into Adam for improved responsiveness.

5.7 SGD at larger scale

Ruder's paper also covers parallelization and distribution schemes like Hogwild, Downpour SGD, delay-tolerant algorithms, and frameworks like TensorFlow that enable efficient distributed training.

5.7.1 Hogwild!

This optimization capitalizes on the inherent parallelism of SGD by allowing multiple threads to update model parameters simultaneously without strict synchronization. This method significantly speeds up training on multi-core processors, effectively leveraging available computational resources.

5.7.2 Downpour SGD

This technique adopts a more structured approach to parallelization by partitioning the dataset across multiple workers, each responsible for computing gradients on a subset of the data. These gradients are then aggregated and used to update the model parameters, enabling more distributed training across clusters of machines.

5.7.3 TensorFlow

Created by Google, this open source framework is well adapted for implementing and deploying large-scale machine learning models. Building on Google's previous work with DistBelief, TensorFlow has found extensive use within Google's internal infrastructure, powering computations across platforms, from mobile devices to distributed systems. Its launch signals a step forward in the field of AI, and hints at a future where machine learning reaches new heights in terms of application.

6 Conclusion

The exploration of gradient descent optimization algorithms provides invaluable insights into the intricacies of training deep neural networks. From the basic SGD to sophisticated adaptations like Adam and AdaMax, where each algorithm has its unique strengths to handle the complexities of optimization.

Momentum and Nesterov accelerated gradient drive convergence and stability, while adaptive methods such as Adagrad, Adadelta, and RMSprop dynamically adjust learning rates to account for different parameter gradients. The discussion also extends to scalable optimization through parallelization and distribution schemes, reflecting the evolving landscape of modern computing.

Additionally, complementary strategies such as curriculum learning, batch normalization, and early stopping enrich the optimization process, promoting efficiency and robustness in model training. In particular, adaptive learning rate techniques such as Adagrad, RMSprop, and Adam stand out for their effectiveness in handling missing data and ensuring convergence in neural network training.

As the field of deep learning continues to evolve, a great understanding and application of these optimization methods will be critical for innovation

and performance. Through the ongoing research and wider implementation, artificial intelligence will undoubtedly be pushed towards new horizons, with adoptions for various applications in all industries.