Classification and regression, from linear and logistic regression to neural network

Johan Mylius Kroken[1,2], and Nanna Bryne[1,2]

[1] Institute of Theoretical Astrophysics (ITA), University of Oslo, Norway
[2] Center for Computing in Science Education (CCSE), University of Oslo, Norway

October 16, 2022 GitHub repo link: https://github.com/Johanmkr/FYS-STK4155colab/tree/main/project2

ABSTRACT

ABSTRACT

****
Felt cute, might delete later:

for ideas (\feltcute)

rephrase this (\rephrase{...})
check if this is correct (\checkthis{...})
* *comment (\comment{...})*
add filler text (\fillertext)
for when you are lost (\wtf)
****

Felt cute, might delete later:

Nomenclature

$\boldsymbol{\beta}$ Parameter vector
C Cost function
γ ...
η ...

* *Should order alphabetically or logically.*

1. Introduction

2. Theory

In linear regression we have the famous assumption that

$$y_i = f(\mathbf{x}_i) + \varepsilon_i \simeq \mathbf{x}_i^\intercal \boldsymbol{\beta} + \varepsilon_i, \tag{1}$$

where f is a continous funtion of $\mathbf{x}$. If we now were to allow said function to represent discrete outputs, we would benefit from moving on to *logsitic* regression.
* *Maybe move to intro?*
add filler text smooth transition to steepest descent

2.1. Stochastic gradient descent (SGD)

Gradient descent (Hjorth-Jensen 2021)
add filler text aim is to find beta etc

2.1.1. Plain gradient descent (GD)

The most basic concept is that of steepest descent. In order to find a minimum of a function $f(\mathbf{x})$ (allow multivariability of $\mathbf{x}$), we follow the steepest descent of that function, i.e. the direction of the negative gradient $-\nabla f(\mathbf{x})$. We thus have an iterative scheme to find minima:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k \nabla f(\mathbf{x}_k), \tag{2}$$

where η_k may be referred to as the step length or learning rate, which
add filler text

What we would like to minimise is the cost function $C(\boldsymbol{\beta})$ which is a function of the parameters $\boldsymbol{\beta}$ which we are trying to estimate. This means that eq. (2) translates into:

$$\begin{aligned} \mathbf{g}_k &\equiv \nabla C(\boldsymbol{\beta}_k) \\ \boldsymbol{\beta}_{k+1} &= \boldsymbol{\beta}_k - \eta_k \mathbf{g}_k \end{aligned} \tag{3}$$

Where we define $\mathbf{g}_k$ to be the direction and magnitude of the steepest ascent in parameter space. For a sufficiently small η_k, this method will converge to a minimum of $\boldsymbol{\beta}$. (However, since we may not know the nature of $\boldsymbol{\beta}$,there is a risk that this is just a local and not a global minimum. The steepest descent method in eq. (3) is a deterministic method, which means we may get stuck in a local minimum. To avoid this we introduce stochastic gradient descent.)

2.1.2. Momentum

From eq. (3) we have that the movement in parameter space is given by $\mathbf{v}_k$ (the negative of which), which describes the direction and magnitude of the steepest ascent in parameter space. Sometimes we might want to move larger distances in one step. This can be achieved by introduction momentum, where we add an addition term to $\mathbf{v}_k$:

$$\begin{aligned} \mathbf{v}_k &= \gamma \mathbf{v}_{k-1} + \eta_k \mathbf{g}_k \\ \boldsymbol{\beta}_{k+1} &= \boldsymbol{\beta}_k - \mathbf{v}_k \end{aligned} \tag{4}$$

where γ is a momentum parameter and η_k is the same learning parameter as before. The basic idea is that this with this method we "overshoot" the descending step length in the direction of the previous step, with a magnitude that is controlled by γ. By doing this, we may reach the desired minimum with fewer iterations.

2.1.3. Stochasticity

There are several weaknesses to the plain gradient descent, perhaps the largest is the computational expense of on large datasets and its sensitivity of initial conditions and learning rates. If $\boldsymbol{\beta}$ have numerous local minima, we will find one minimum only per set of initial conditions, and we have no good way of saying whether this minimum is global or not. One way of overcoming this is by adding stochasticity to the gradient descent algorithm.

The main idea is that we have n data points, that we divide into M minibatches (subsets), meaning that we have n/M data points in each minibatch, denoted B_j for $j \in [1, 2, \ldots, n/M]$. We also recognize that we may write the total cost function as a sum over all data points $\mathbf{x}_i$ for $i \in [1, n]$, and thus approximate the gradient of the cost function by only summing over the data points in a minibatch picked at random:

$$
\begin{aligned}
C(\boldsymbol{\beta}) &= \sum_{i=1}^{n} c_i(\mathbf{x}_i, \boldsymbol{\beta}) \\
\nabla_\beta C(\boldsymbol{\beta}) &= \sum_{i=1}^{n} \nabla_\beta c_i(\mathbf{x}_i, \boldsymbol{\beta}) \\
\tilde{\mathbf{g}}_k &= \sum_{i \in B_j} \nabla_\beta c_i(\mathbf{x}_i, \boldsymbol{\beta}_k)
\end{aligned} \tag{5}
$$

2.1.4. Tuning

* *Something about hyper parameters λ and η*

2.2. Neural Network (NN)

2.2.1. Basics

2.2.2. Activation functions

2.2.3. Back propagation

2.3. Classification

2.4. Logistic regression

3. Conclusion

Code availability

The code is available on GitHub at https://github.com/Johanmkr/FYS-STK4155colab/tree/main/project2.

References

Hjorth-Jensen, M. 2021, Applied Data Analysis and Machine Learning (Jupyter Book)