FYS-STK4155
Applied data analysis and machine learning
Project 3

Henry Haugsten Hansen , Fred Marcus John Silverberg

14 December 2018

Abstract

Contents

* The abstract should contain concrete results

1 Introduction

* Status of problem * Major objectives * Motivation for the reader for why they should care about the problem * Motivation for why we consider these models in particular

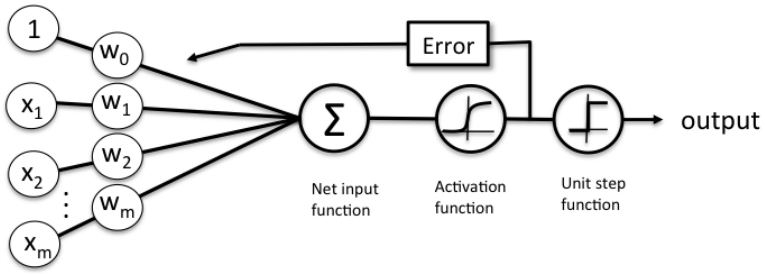
2 Theory

2.1 Election data

2.2 Logistic regression

Logistic regression is primary used for classification problems, where the target is a discrete variable. The method measure the relationship between the dependent variable and the independent variables by estimating probabilities. This is possible due to the logistic function, which transform the relationship into a number between (0,1). This number can in turn be transformed into a binary value by a threshold classifier, which determine if the value is high enough for a certain class. A figure is included below that illustrate the method.

Figure 1: Logistic regression model



Source: `https://rasbt.github.io/mlxtend/user_guide/classifier/LogisticRegression/`

By initializing random and normal distributed weights W_i the method multiplies them with the input variables X_i, which are then summed and activated by the logistic function, introducing non-linearity into the model. An error measurement is done and used for adjusting the weights. This process is repeated and finally the resulting probabilities goes through the threshold classifier for a final output. Imagine a forward and backward calculation encapsulated in an iteration process, then it can be further described as follows:

Cost function
The model is assigned a suitable cost function, which is to be minimized throughout the process. In this project the quadratic is for prediction and the cross entropy for classification.
Quadratic:

$$Cost(\hat{W}) = \frac{1}{m}\sum_{i=1}^{m}(yp - y)^2 \tag{1}$$

Cross Entropy:

$$Cost(\hat{W}) = \frac{-1}{m}\sum_{i=1}^{m}(y_i * ln(yp) + (1 - y_i) * ln(1 - yp)) \tag{2}$$

Where y is the target vector holding observed values, while yp is a function of $f(\hat{X}, \hat{W})$ and represents the predicted output vector, both with elements represented by i.

Forward phase
With a cost function defined, the first step is to calculate the net input value of the randomized weights assigned, where W_o can be thought of as the bias b.

$$Z = netinput = \sum_{i=1}^{m} X_i \bullet W_i + b \tag{3}$$

The net input values are then put into the activation function, obtaining probability values. There is many functions to choose from, in this project the standard will be the sigmoid function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

Back-propagation phase
An error calculation is made and used by the model to adjust the weights. This is usually done by using a gradient decent method. The chosen method here is the Stochastic Gradient Decent (SGD) with mini batches, which allows us to broadly search for the minima of our cost function. Instead of taking small but accurate steps towards the minima, the SGD rushes and stumbles across the cost function until it finds the treasure. This method decrease the computational time, which could otherwise be very heavy, as well as decrease the probability of obtaining a local minima instead of the global minima. For the first iteration, the gradient will to some accuracy, be in the direction of the steepest descent.[**?**]

With cross entropy chosen as cost function, the new weight value will be:

$$W' = W - \eta * \frac{\partial CW}{\partial W} \tag{5}$$

Where η is the regularization term, decreasing the probability of over-fitting. By using the chain rule, the partial derivative is written as:

$$\frac{\partial C(W)}{\partial W} = \frac{\partial C(W)}{\partial yp} * \frac{\partial C(yp)}{\partial Z} * \frac{\partial C(Z)}{\partial W} \tag{6}$$

Expanding the first term as the derivative of our cost function w.r.t prediction:

$$\frac{\partial C(W)}{\partial yp} = -\frac{y}{yp} + \frac{1-y}{1-yp} = \frac{yp - y}{yp(1-yp)} \tag{7}$$

Expanding the second term as the derivative of prediction, which is the same as the activation function of (Z) w.r.t (Z) :

$$\frac{\partial C(yp)}{\partial Z} = activationfunction(Z)' \tag{8}$$

For Sigmoid that is:

$$\frac{\partial C(yp)}{\partial Z} = yp(1-yp) \tag{9}$$

Expanding the third term as the derivative of net input (Z) w.r.t weights:

$$\frac{\partial C(Z)}{\partial W} = X \tag{10}$$

Calculating each partial derivative and simplify it results in:

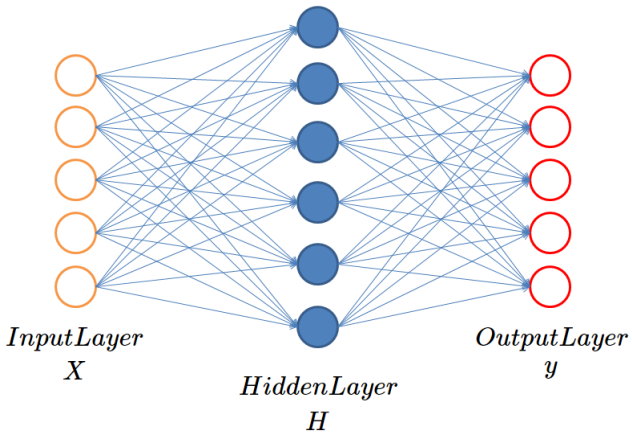
$$\frac{\partial C(W)}{\partial W} = (yp - y) * X \tag{11}$$

Adding the regularization term avoiding over-fitting: $W = W * (-\eta)$
The updated weights are then:

$$W' = W * (yp - y) * X \tag{12}$$

2.3 Neural network

Figure 2: Logistic regression model



Source: `https://chunml.github.io/ChunML.github.io/project/Creating-Text-Generator-Using-Recurrent-Neural-Network/`

A neural network is constructed as shown above. Here the output layer contains more than one output, if wanted. And one hidden layer is added, constructing a net of weights (blue lines) and biases. Additional hidden layers can be included. Each hidden layer can be assigned arbitrary many neurons (the blue circles). The idea is sprung from the logistic regression and extended into a more complex configuration of interconnected elements that processes the information dynamically. Further break down of the concept is as follows, with the encapsulated iteration process in mind:

A suitable cost function is assigned to the model.

Forward phase
Weights and biases are created randomly and normally distributed. The net input value (Z) is then calculated as:

$$Z = X \bullet W + b \tag{13}$$

Noted is that the calculations now take place as nestled matrices, where the matrix W represent all weights on both sides of a calculated layer, hence have the size (2,).

Clarity:
For figure (3): $W(0)_{ij}$ is the matrix of size (Inputs,Neurons) where i denote the input X_i and j denote the neuron H_j, hence holding all weights between the input layer and the hidden layer. While $W(1)_{jk}$ is the matrix of size (Neurons,Outputs) where j denote the neuron H_j and k the output O_k. Which corresponds to the weights between the hidden layer and the output layer. Similary for the bias b, which is a matrix (2,) that hold vectors of biases for each

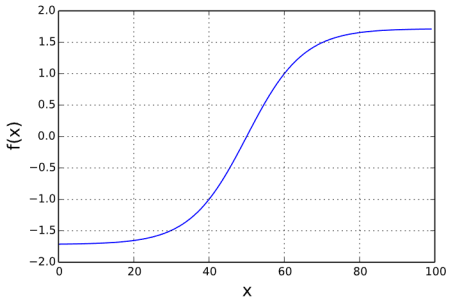
layer. $b(0)$ hold the biases for the hidden neurons and $b(1)$ the biases for the outputs

Z is then activated by chosen activation function for the interaction between the input layer and the hidden layer, and similarly by the hidden and output layer.

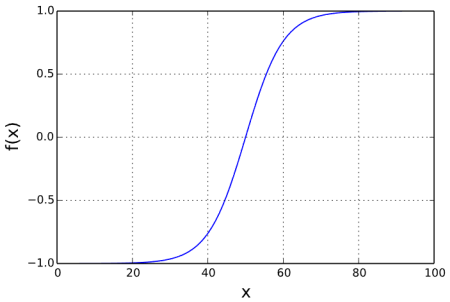
$$a(Z) \tag{14}$$

Figure 3: Activation functions

(a) Sigmoid.



(b) tanh.



Back-propagation phase
Here the golden mathematics take place. First an error estimate is drawn δ_3, which is then passed back by the back-propagation algorithm for adjustments of the weights and biases. The gradients are derived with the same analogy as in the section of logistic regression with a stochastic gradient decent method. For cross entropy as choice of cost function and Sigmoid as activation function for the output layer, the back-propagation algorithm is implemented as below:[**?**]

$$\delta_3 = (yp - y) \tag{15}$$

$$\delta_2 = \delta_3 \bullet W_2^T * (a(Z_1))' \tag{16}$$

$$\frac{\partial C}{\partial W_2} = a(Z)_2^T \bullet \delta_3 \tag{17}$$

$$\frac{\partial C}{\partial b_2} = sum(\delta_3) \tag{18}$$

$$\frac{\partial C}{\partial W_1} = X^T \bullet \delta_2 \tag{19}$$

$$\frac{\partial C}{\partial b_1} = sum(\delta_2) \tag{20}$$

Layers are representative by (1,2,3) = (I,H_1,O) such that the recipe can be extended if additional hidden layers are added, as (1,2,3,4) = (I,H_1,H_2,O). Note that if the cost function or the activation function are changed, one would have to derive δ_3 from scratch.

A pseudo code of the Neural Network with SGD:

```
for i in range(epochs):
   * Construct mini batches
   for j in range(batches):
      * Draw mini-batch
      * Forward phase
         (Activates all neurons and predict outputs)
      * Back-prop phase
         (Estimate error)
         (Calculate gradients)
         (Update weights and biases with regularization)
Final prediction or classification
```

There must be highlighted that the neural network are prone to some parameters that governs entirely how well the model will preform.[**?**]

Learning rate
A hyper-parameter that controls how much the weights and biases will adjust due to the gradient of the cost function. A small value could be described as taking small but accurate steps towards the minimum. For a large value, the path towards the minimum would be rough. To large and there is a probability of never reaching the precise minima, while to low may be stuck at a plateau. It also affects the computational time, a large learning rate would decrease the computational time and vice verse.

Regularization term
By applying a regularization term, the weights and biases (or coefficients for the linear case) is basically programmed to be small, with the assumption that a lower variance between them would represent a simpler model, hence it is a way of avoiding over-fitting.

Epochs
Denotes how many times a training set is run through the neural network, both forward and backward. By increasing the number of epochs one would expect the model to obtain optimal weights and biases, only running it once would under-fit the model.

Batch size
Included in the SGD method, the training data is divided into smaller batches which run separately through the neural network. So instead of running the whole training set in one epoch, all batches runs once for one epoch. This allows us to run larger data sets. Optimal batch size is discussed but the size will effect the convergence of the gradient. Generally a small batch size in the range of 32-512 is to prefer.[**?**]

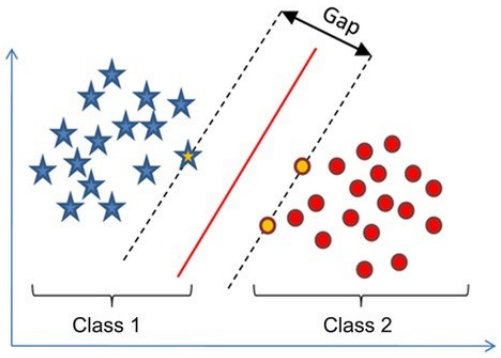
Layers and Neurons
Increasing the number of neurons or layers also increase the networks ability to learn detailed parts of the data, so for a complex situation more neurons and / or layers would be necessary. This can be adjusted along the way, by doing a grid-search for different numbers and compare the final score. The configuration also have impact on the proper choice of activation function.

Activation function
The choice of activation function will have impact on how the model preform as it transform the values into different domains. They may have steeper gradients and lower computational time, or be able to avoid vanishing gradients allowing deeper nets, or decrease the probability of exploding cost functions. For a network with multiple layers, one could use different functions for each layer.

2.4 Support Vector Machine

A machine learning method that is broadly used in classification problems, noted is that it could as well be used for regression purpose. It relies on supervised data to construct a decision boundary that divides classes in space. The advantage of the algorithm lies partly in the support vectors, marked as dotted lines in figure[4]. The optimal decision boundary (red line) is obtained by maximizing the gap between the support vectors. This feature is highly appreciable when dealing with classes that are difficult to separate. And further advantage is gained by the allowance of analysing data in higher dimensions.

Figure 4: Visualized SVM concept



The decision boundary and support vectors are here described by:

$$y = w^T x + b = 0 \qquad , \qquad y = w^T x + b = \pm 1 \tag{21}$$

Where [w] is the normal vector to the decision boundary, [b] is the bias. The maximal margin, or gap in figure[4] is defined as:

$$max_w \frac{2}{||w||} \qquad or\ equivalently \qquad min_w ||w||^2 \tag{22}$$

Cost function:
Mentioned margin is to be optimized. But first we introduce a slack variable [ξ], which for [$\xi \geq ||w||$] a point is misclassified and for [$0 < \xi \leq ||w||$] a point is located within the margin. This is added as a term which operate as a margin softener, simply allowing some mistakes, we have:

$$min_w ||w||^2 + C \sum_{i=1}^{m} \xi_i \quad constrained\ to \quad y_i * f(x_i) \geq 1 - \xi_i \tag{23}$$

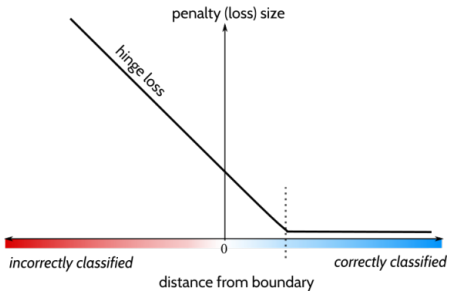
Here C is the regularization parameter that regulates how much we value the mistakes and [x_i] is the support vectors. By including [$\xi_i \geq 0$] in the constraint, we can write the constraint and optimization function respectively as:

$$[\ y_i * f(x_i) \geq 1 - \xi_i\ ] \rightarrow [\ \xi_i = max(0, 1 - y_i * f(x_i))\ ] \tag{24}$$

$$min_w ||w||^2 + C \sum_{i=1}^{m} max(0, 1 - y_i * f(x_i)) \tag{25}$$

Where [$f(x_i) = w^T x_i + b$]. The summation term is denoted the hinge loss function, illustrated in figure[5].

Figure 5: The hinge loss function



Since this a non smooth function, it is needed to interpret it as a piece-wise function in order to minimize it. We can denote the term inside the summation as: [$max(0, 1 - y_i * f(x_i)) = L(x_i, y_i, w)$], by observing the two possibilities:

$$\frac{\partial(L)}{\partial(w)} = -y_i x_i \quad , \quad \frac{\partial(L)}{\partial(w)} = 0 \tag{26}$$

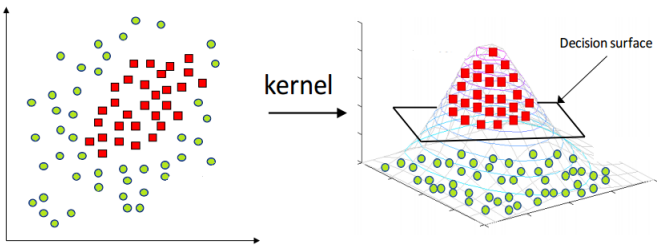
Further, we take use of the Lagrange method and define $\lambda = 2/(mC)$. We can now write the cost function to minimize as:

$$Cost(w) = \frac{1}{m} \sum_{i=1}^{m} (\frac{\lambda}{2} ||w||^2 + L(x_i, y_i, w)) \tag{27}$$

Kernel trick

By transforming the above primal form into dual form, the advantage of using the kernel-trick become available. The trick essentially transform a non-separable space into a separable, allowing us to obtain a decision boundary.

Figure 6: Kernel trick visualized



This is done by observing that the solution [w] can be expressed as a linear combination of the data:

$$w = \sum_{k=1}^{m} = \lambda_k, y_k, x_k \tag{28}$$

This is then substituted into $f(x_i)$ and the defined cost function, leading to a optimization problem over λ instead of w, with the following decision function:

$$f(x) = \sum_{i=1}^{m} \lambda_i y_i K(x_i, x) + b \tag{29}$$

And the optimization function follows as:

$$max_{(\lambda_i \geq 0)} \ \ \lambda_i - \frac{1}{2} \sum_{kj} \lambda_k \lambda_j y_k y_j K(x_k, x_j) \ for \ [0 \geq \lambda \geq C] \ and \ \sum_{i=1}^{m} \lambda_i y_i = 0 \tag{30}$$

Finally the kernel trick can be applied by inserting chosen kernel function suitable for the specific classification problem. A few commonly functions are:

Linear

$$K(x_i, x) = x^T x_i \tag{31}$$

Polynomial

$$K(x_i, x) = (x^T x_i + \gamma)^d \tag{32}$$

Radial basis

$$K(x_i, x) = e^{-\gamma ||x - x_i||^2} \tag{33}$$

Parameters:
For being able to obtain the most efficient model of data, one need to tune some parameters. There is no golden rule here, the tuning must be done by consideration for each applied situation. This is usually handeled by doing a cross validation search. In addition to the parameters described below the different kernel functions also govern the final results, hence experimanting with those is also recommended. [4]

C
A parameter that governs the sensitivity of correct classifications. It is essentially a trade-off between the amount of misclassification on training data and the maximization of the margin between the support vectors. For larger values of C only smaller margins would be allowed and for smaller values the margins would be larger and hence the decision boundary would be simpler, at the expense of accuracy. For $C = \infty$ the margin is denoted hard and do now allow any misclassification's. C is simply the regularization factor for SVM.

γ
This parameter is part of the kernel function and vary the dependence on whether the algorithm will construct a decision boundary with high consideration of points in its neighbourhood, or not. A low value would loosely fit the data, hence under-fitting it with a less complex boundary, and vice verse for a high gamma value.

2.5 Model assessment

[Below, y represents the target while yp represents the prediction.]

Mean square error:
MSE is the average of square errors, the difference between the observed values and the fitted values. Each difference is squared such that negative and positive

values can be handled without interfering. A small value is preferred, since MSE = 0 is regarded as a perfect fit. Noted is that $MSE = Bias^2 + Variance + Noise$

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - yp_i)^2 \tag{34}$$

R2-Score:
Describe the total variance by the fitted values divided by the total variance of the observed values. Hence giving a picture of how well the fitted model correlates to the observed. The value spans between [-inf,1] where 1 represents perfect correlation.

$$R^2 = 1 - \frac{\sum_{i=0}^{n-1}(y_i - yp_i)^2}{\sum_{i=0}^{n-1}(y_i - E[yp])^2} \tag{35}$$

Accuracy Score:
This measure the number of correct classifications. Which is used for evaluating the network. If all elements are classified correctly the score will be 100 %.

$$A\text{score} = \frac{1}{n}\sum_{i=1}^{n} I * (y_i = yp_i) \tag{36}$$

Where I is the indicator function, 1 if $y_i = yp_i$ or 0 if $y_i \neq yp_i$.

Bias:
Bias is a measurement of the difference between the actual target value and the predicted value for a specific x input. It may give indication whether or not a model is over-fitted or under-fitted. A high bias value indicates the later.

$$Bias^2 = \frac{1}{n}\sum_{i=1}^{n}(E[yp] - y_i)^2 \tag{37}$$

Note that the expectation (E) values are for different sets $[x0, x1, x2...xn]$,$[y0, y1, y2...yn]$ out of the training set: $P(x, y)$. Where yp and y are functions of x.

Variance:
Variance measure the difference between the average predicted value and each single predicted value $yp_i = f(x_i)$ out of the set of x inputs. It describe the spread between the output values, a high value may indicate an over-fitted model and vice verse.

$$Variance = \frac{1}{n}\sum_{i=1}^{n} E[yp_i - E[yp]]^2 \tag{38}$$

Resampling:
A tool that is used on limited data. It allows the analyser to draw more information out of the data available, hence evaluating the errors further and identify the variability without calculating the covariance. It is computationally heavy since the idea is to repeatedly train on one piece of data while testing on the left out piece. Bootstrap is here the chosen among many techniques. The concept is to randomly draw new datasets from the training data, with replacements.

Then preform a new prediction and new statistics. This is done n times and is later averaged. The results may indicated whether or not the model used is to complex (high variance in the training set) or to simple (high bias in the training set).

3 Code Implementation

4 Results

4.1 LOG

4.2 SVM

4.3 NN

5 Discussion

6 Conclusions

References

[1] Nasa Horizon. 2018. HORIZONS Web-Interface. [ONLINE] Available at: http://ssd.jpl.nasa.gov/horizons.cgi. [Accessed 24 October 2018].

[2] Morten Hjorth-Jensen. 2018. Computational Physics Lectures: Ordinary differential equations. [ONLINE] Available at: http://compphysics.github.io/ComputationalPhysics/doc/pub/ode/html/ode.html. [Accessed 24 October 2018].

[3] Ritchie Ng. 2018. Support Vector Machines [ONLINE] Available at: https://www.ritchieng.com/machine-learning-svms-support-vector-machines/ [Accessed 14 December 2018].

[4] $Scikit-learn$. 2018. RBF SVM parameters. [ONLINE] Available at: $https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html$ [Accessed 14 December 2018].

Access to all material can be found at: https://github.com/silverberg89/FYS4150/tree/master/Project3