Project 3 Wisconsin Cancer data and Logistic regression and Feed Forward Neural Networks

Annika Eriksen

December 17, 2020

Abstract

The wisconsin cancer data is a staple of machine learning and should therefore be a good point to analyse different methods. I analyze the data using 2 different methods, firstly through logistic regression and later on using a Feed Forward Neural Network

1 introduction

The Wisconsin Cancer dataset is a widely used set for classification problems, and offers a good starting point to compare implementations of machine learning methods. I have put the set through 2 different methods, Logistic regression and a Feed Forward Neural Network.

First, the Logistic regression takes in the data and feeds it into the sklearn logistic regression method. This is done without- and then with scaling the data to see prediction accuracy depending on these. Then I take only those features with the greatest correlation to a malignant tumour and train the set on those only, to see the predictive ability of the methods based on these.

While we see an expected increase in accuracy after normalising the features, there is a rather unexpected decrease in accuracy when only including the more important features.

Then, Implementing a Feed Forward Neural Network (ffnn), We construct a network with 1 hidden layer and a binary input, either malignant or benign. The initial model is then iterated a number of times, or until the predictive accuracy of the model on the test set has passed it's peak.

As one expects, the neural net quickly goes toward an above 90% accuracy for both training and test set until eventually overfitting sets in and the test set accuracy starts to fall off.

2 methods

2.1 Logistic Regression

The base problem to be solved with the cancer data set is a binary benign vs. malignant tumor, 0 or 1 encoded respectively,

$$y = \begin{bmatrix} 0 & \text{benign} \\ 1 & \text{malignant} \end{bmatrix}. \tag{1}$$

We have 2 *classes*, which form the entirety of the possible outcomes within the set. Like with regression I want a model to give an expected value for a given set of samples for the features. A way to model would be to assume some linear functional dependence, along the lines of (Jensen (2020))

$$f(y_i|x_i) = \beta_0 + \beta_1 x_1.$$

Although this could contain values outside of 0 and 1. Taking the mean solves this and forces $f(x|y) \in [0,1]$. This then can be seen as the probability for finding a given value for y_i with a given x_i. This S-shaped function would give us the *sigmoid* function. The soft predictor gives a probabilistic prediction of the class of outcome, given the input feature. In my case, this is either of the options in y in eq 1. The Sigmoid, or *logit* function governs this probability prediction for a given event thus, (Jensen (2020))

$$p(t) = \frac{1}{1+\exp(-t)} = \frac{\exp(t)}{1+\exp(t)}. \tag{2}$$

Noting, $1 - p(t) = p(-t)$.

Since we have 2 possible outcomes, the probabilites are then given by

$$p(y=1|x_i,\hat{\beta}) = \frac{\exp(\beta_0 + \beta_1 x_i)}{1+\exp(\beta_0+\beta_1 x_i)} \tag{3}$$

$$p(y=0|x_i,\hat{\beta}) = 1 - p(y=1|x_i,\hat{\beta}). \tag{4}$$

Denoting the biases with **fi** which I want to extract from the dataset.

Using the Maximum Likelihood Estimation principle The total probability for all possible outcomes of a dataset $\mathcal{D} = (y,\mathbf{x_i})$, with the binary in eq. 1 for each y_i. The probabilities can be approximated by a product of the individual probabilities for an outcome of y_i,

$$\mathbf{P}(\mathcal{D}|\hat{\beta}) = \prod_{i=1}^{n} \left[p(y_i=1|\hat{\beta})\right]^{y_i} \left[1 - p(y_i=1|\hat{\beta})\right]^{1-y_i} \tag{5}$$

And from this we can find the log-likelihood and our *loss* function.

$$\mathcal{C}(\hat{\beta}) = \sum_{i=1}^{n} \left(y_i \log \left[p(y_i = 1 | x_i \hat{\beta}) \right] + (1 - y_i) \log \left[1 - p(y_i = 1 | x_i \hat{\beta}) \right] \right). \tag{6}$$

Inserting for the probabilities with eq. 3 and sorting the logarithms, we can rewrite the loss function. The cost/error function is the negative log-likelihood function, so it would be the negative of this rewrite. The end result then, is

$$\mathcal{C}(\hat{\beta}) = -\sum_{i=1}^{n} \left(y_i(\beta_0 + \beta_1 x_i) - \log(1 + \exp(\beta_0 + \beta_1 x_i)) \right). \tag{7}$$

This is also known as the cross-entropy. This can be further supplemented with regularization parameters much like with Lasso and Ridge regression.

The next step then is to minimize the cost function. Differentiating wrt. β_0 and β_1 and compacting, gives us

$$\frac{\partial \mathcal{C}(\hat{\beta})}{\partial \hat{\beta}} = -\hat{\mathbf{X}}^T (\hat{y} - \hat{p}). \tag{8}$$

Where $\hat{y}$ is a vector with the values y_i, $\hat{\mathbf{X}}$ is an $n \times p$ matrix containing the x_i values. The vector $\hat{p}$ contains the fitted probabilities $p(y_i | x_i, \hat{\beta})$. For a step further and introducing another matrix, $\hat{\mathbf{W}}$ with elements $p(y_i | x_i, \hat{\beta})(1 - p(y_i | x_i, \hat{\beta})$, we can arrive at a more compact version of the 2. derivative.

$$\frac{\partial^2 \mathcal{C}(\hat{\beta})}{\partial \hat{\beta} \partial \hat{\beta}^T} = \hat{\mathbf{X}}^T \hat{\mathbf{W}} \hat{\mathbf{X}}. \tag{9}$$

With the regression model accounted for, the script itself imports and uses the *scikitlearn* module in python, where the data is loaded from, scaled and split into training and testing sets in various permutations. The regression itself is mainly done with scikitlearn's *LogisticRegression* method, implementing the mathematics above with a well documented and tested code. It is also optimized well beyond the possible approach of this project. It is therefore an easy to use tool to check the data.

Once the data set is loaded and some parts are displayed, 3 different regressions are made. 1 for the raw set, one where the feature data is scaled and one where a percentage of the most important features are selected for and trained on.

2.2 Feed Forward Neural Net

A neural network is a computational model which aims to imitate the workings of biological neurons, where an arbitrary amount of layers, each with an arbitrary number of neurons exchange signals in the form of mathematical functions. These systems take in example inputs and configures an output without any specific rules for a given task

The basic idea is that each neuron collects incoming signals. If the signals exceed a given activation threshold, the neuron fires as well. the signals failing to meet this threshold leave the neuron inert and there is no signal. A simple mathematical model of this can be written,

$$\mathbf{y} = \mathbf{f}\left(\sum_{i=1}^{n} \omega_i x_i\right) = \mathbf{f}(\mathbf{u}). \tag{10}$$

Where the output y is the output of the activation function and takes the input of a weighted sum of n other neurons. Though there are various types, e.g. Convolutional neural networks for image recognition, The focus for this will be a simple general purpose one for supervised learning, seeing as the cancer data is already properly labeled.

The model used in this project is a Feed forward Neural network. This is the first type developed and the simplest type. Here, the information flows only one way. Each node in a layer is connected to every other node in the next layer, mediated with weights. Therefore called a *Fully Connected FFNN*

A neural net with 3 or more layers, an input and output layer with one or more hidden layers between, is frequently called a *Multilayer Perceptron (MLP)*. These are utilized because the *universal approximation theorem* shows us that Given a non-constant, bounded and monotonically increasing activation function for the hidden layer. Given a non-constant, bounded and monotonically increasing activation, an MLP can predict a continuous multidimensional function to arbitrary accuracy with only 1 hidden layer and a finite set of neurons.

The output is then given by the function

$$y = f\left(\sum_{i=1}^{n} (w_i x_i + b_i)\right) = f(z). \tag{11}$$

This is similar to equation 10, but with an added intercept per node. The MLP is fully connected, as stated above. Thus each node receives a weighted sum of the output of every other node in the previous layer.

For the first hidden layer, each node z_i^1 is then a weighted sum of the outputs of the nodes int the input layer.

$$z_i^1 = \sum_{j=1}^{N} w_{ij}^1 x_j + b_i^1. \tag{12}$$

The b_i^1 is the bias for the node, which is needed in the case of a 0 input. The value of this sum is the input into the activation function, eq. 11, for the i-th node. The summation is over all N possible inputs into the first layer.

More generally, we can assume that the nodes within a layer share an activation function but that the activation functions could vary between layers. This can be shown with a superscript l.

$$y_i^l = f(z_i^l) = f\left(\sum_{j=1}^{N_{l-1}} (w_{ij}^l x_i^l) + b_i^l\right). \tag{13}$$

Here, the N_{l-1} denotes the number of nodes on the previous layer, as the M in equation 12 denotes the number of nodes in the input layer. This process can be repeated through the layers, Feeding Forward to the output layer. This expression eventually becomes then a sum of the sum of the sums of the previous layers. In essence, then the only independent variables in the system is the initial values. The rest inductively follow.

With the input settled, the question of activation function comes next. As mentioned wrt. the universal approximation theorem , the activation function has a few requirements. The activation function must be non-constant, bounded, monotonically-increasing and continuous. The second option here excludes any linear function. In order to avoid the neural net producing a linear transformation through the layers, some non-linearity is needed. A common activation function is the *sigmoid function*

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{14}$$

There are many other options, but they are not explored with this project. There are some issues with the sigmoid function wrt. gradient descent methods as the gradients can tend to disappear for the sigmoid, despite it's initial popularity due to the similarity with the potential curves within the biological neurons. A more popular function is the tanh function.

With the layers and nodes described as well as the signal connecting them from input to output, There remains an issue of predictive ability. Sending the data through and setting the weights the once is not the best basis for such things. To improve the results, we can adjust the weights once we've found an output. The method for this is the *back propagation algorithm*

The back propagation error is

$$\delta_j^l = \sum_k \quad \delta_k^{l+1} \quad w_{kj}^{l+1} \quad f'(z_j^l) \tag{15}$$

Which we apply once the layers have been passed through forward, using the outputs of each layer and propagating the errors backwards. Using gradient descent, we update the weights of each node according to the gradient and the learningrate. The source code for both methods can be found at the projects github page, Eriksen (2020).

3 results

The results for the Logisti regression case provides a good approximation of the data, where the naive implementation on the data without scaling it predicts

Full set, unscaled	0.96
Full set, scaled	0.98
exclusive features, scaled	0.95

Table 1: Logistic regression scores for Wisconsin Cancer data. The unscaled data provides a predictive accuracy of above 95%, with the scaled data gets as far as 98%. The final selection uses only those features of the data that have the best correlation with a malignant and benign tumor. Here the accuracy score drops

with an accuracy of 96%. Scaling the data provides a better predictive accuracy by culling outlying data, bringing the accuracy of the model up to 98%. Finally with the regression case, removing the less important data and training only on those provides a poorer predictive accuracy than the 2 sister methods at 95%, see table 1

For the neural network the accuracy of the prediction quickly climbs to 94% accuracy for both the training and test set after fewer than 10 epochs. Then the growth mellows somewhat. The training accuracy grows to about 99% and remains stable here, the test accuracy climbs to a maximum of 97% after 200 epochs. The test accuracy eventually begins to taper off again towards the end. After roughly 600 epochs, the system is as accurate at predicting other data as it can be. Continued training at this point, as the predictive accuracy falters. The 2 accuracies can be seen in figure 1

4 conclusion

The results from the logistic regression points towards predicting based on all features rather than stripping away the less relevant features. This seems counterintuitive. A more likely outcome is an error in either the selection of relevant features, based on the correlation between the feature in question and the output values, or in the implementation of the scoring values.

As for the neural network, the accuracy plot follows the expected pattern, both approaching better accuracy with the training of the network, while eventually reaching the region of over-fitting where the accuracy of the test set drops off over time.

In this data set the 2 main methods produce very similar results, with the best predictive case eking out a narrow lead in terms of predictive ability wrt. to the neural net's maximum test accuracy of 97% to the logistic regression's 98%.

This is a very limited study and part of the results seem to deviate from the expected trajectory of prediction in the logistic regression case, as removing the noise of the less relevant features should increase accuracy of prediction, because the model didn't have to try and fit this noise.

Further work would definitely include a deeper study of the logistic regres-

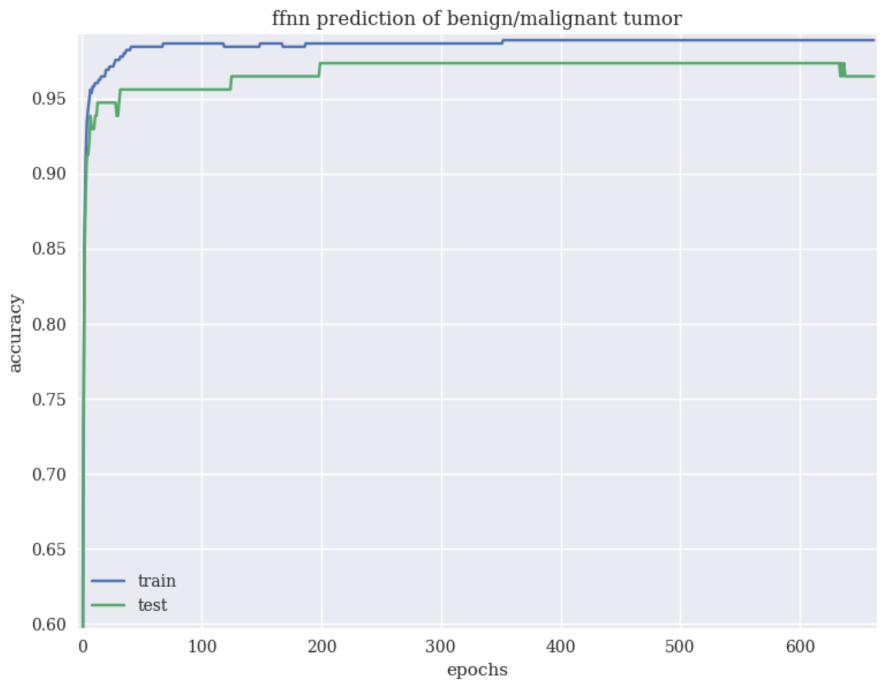

Figure 1: Plot of the accuracy of prediction of the neural net for the Wisconsin cancer data set. The accuracy is the ratio of accurate predictions over the different epochs. The blue training accuracy quickly goes to 0.99, while the test sett accuracy peaks some 150 epochs later at 0.97 before lowering gradually after about 600 epochs.

sion case. Figuring out and controlling for the feature selection process, and exploring different cos functions. In the neural net case, further exploration could be in whether the number of layers would impact the prediction. There are also variables wrt. the activation and back-propagation algorithms used.

References

[Eriksen 2020] ERIKSEN, Annika: *project 3 sourcecode*. 2020. – https://github.com/ageriksen/fysstk4155/tree/master/project3

[Jensen 2020] JENSEN, Morten H.: *lecturenotes fys-stk4155*. 2020. – https://compphysics.github.io/MachineLearning/doc/web/course.html