Project 3 FYS-STK4155

Reinert Eldøy

December 18, 2022

Abstract

Three different models are trained on a breast cancer dataset, aiming to correctly classify tumor samples as benign or malignant. The models are logistic regression, a feedforward neural network, and random forests.

Introduction

In this project I will compare the performance of a logistic regressor, a feed-forward neural network, and a random forest classifier on the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset. This contains data about tumor tissue samples as well as whether or not they're malignant.

Data

The dataset consists of 569 samples with 30 features that are named

```
['mean radius', 'mean texture', 'mean perimeter', 'mean area',
   'mean smoothness', 'mean compactness', 'mean concavity',
   'mean concave points', 'mean symmetry', 'mean fractal dimension',
   'radius error', 'texture error', 'perimeter error', 'area error',
   'smoothness error', 'compactness error', 'concavity error',
   'concave points error', 'symmetry error',
   'fractal dimension error', 'worst radius', 'worst texture',
   'worst perimeter', 'worst area', 'worst smoothness',
   'worst compactness', 'worst concavity', 'worst concave points',
   'worst symmetry', 'worst fractal dimension']
```

These relate to various physical features of tumors. The target dataset however is a boolean array where 0 represents benignancy and 1 malignancy.

Methods & Theory

Logistic regression

This is a regression method primarily used for classification tasks. It could be as simple as a spam filter accepting or rejecting an email, a situation we might want to represent as $y \in \{0, 1\}$. This isn't continuous and in practice it turns out to be better to assign probabilities to either choice. Such a probability function could be the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

which looks like in Fig 1. Its smoothness is of immediate note. For purposes of machine learning we can instead consider the conditional probability

$$P(y_j = 1|x, \theta) = \frac{1}{1 + e^{-x_j^T \theta}} = \sigma(x_j^T \theta)$$

which is to be interpreted as the probability of observing that $y = 1$ *contingent on* the model parameters θ. So what's a good cost function for logistic regression? The answer is the *cross entropy*

$$C(\theta) = \sum_{i=1}^{n} -y_i \log(\sigma(x_i^T \theta)) - (1 - y_i) \log(1 - \sigma(x_i^T \theta))$$

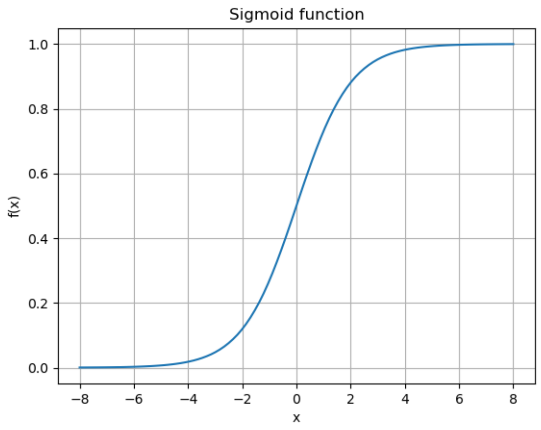


Figure 1: The sigmoid function appears like a smoothed out step function

Neural Networks

The Perceptron

Artificial neural networks are inspired by the electrochemical operation of biological neural networks, or brains if you will. The fundamental unit of this system is called a *perceptron* and consists(see figure below) of an *input layer* connected to a central node by weighted edges. The inputs and weights are

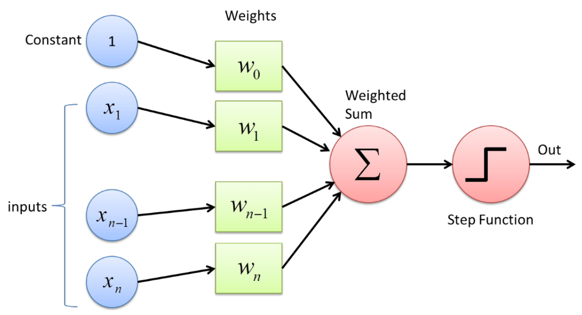


Figure 2: Taken from https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

then combined in the algebraic expression

$$z = \sum_{i=1}^{n} w_i x^i + b_i$$

where I've included an important bias term. Next up we apply a nonlinear *activation function* to the expression. In the case of the perceptron this is the Heaviside step function, but another popular choice is the *sigmoid* function,

$$\sigma(z) = \frac{1}{1 + e^{-x}}$$

or the *ReLU* function

$$ReLU(x) = \max{(x, 0)}$$

Thus the output of the perceptron looks like $\sigma(\sum_{i=1}^{n} w_i x^i + b_i)$. This model is meant to mimic a single biological neuron such that the input edges might represent signal receiving dendrites and the output edge might represent the signal transmitting axon. But a biological network may consist of billions of neurons working in concert, and in the same way multiple perceptrons can tie together in large neural network architectures. This project will concern itself with one such architecture.

The Feedforward Neural Network

A feedforward neural network(henceforth "FFNN") consists as before of an input layer and an output layer, but now the output layer could contain multiple nodes rather than just the one "nucleus" in the perceptron. In addition we could have a layer in between the two known as a *hidden layer*, in fact we could have multiple of these with as many hidden neurons as we please. To top it all, as we scan from left to right we see that each node in a layer to the left is connected

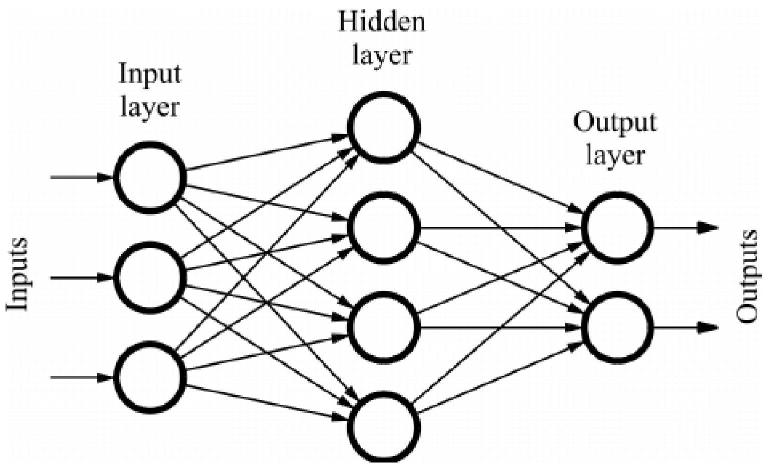


Figure 3: From https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

to *every* node to the right. The final characteristic of the FFNN is that no edge loops backwards, hence "feedforward". As before each edge is weighted with a number denoted w^l_{jk} where l labels the layer it's pointing into and it points from neuron j to neuron k. Not shown in Fig. 2 is the possibility(or even necessity) of a bias term b^l_j at any of the layers. In addition to the structural freedom, we are free to pick any activation function we please at any of the neurons, and so this model seems to have a great degree of inherent flexibility.

Training the network

There are various and sundry potential applications for neural networks, from linear regression to interpreting handwriting. Given a dataset to be fed through the input layer, how can we adjust the weights so that the output at the other end holds true to this promise? The key is gradient descent. Recall that the cost function is defined on a landscape of tunable coefficients, or weights. With sufficiently large FFNNs it would be futile to compute the full gradient, so SGD is the natural choice for optimizing the model. Computing the gradient directly would still be difficult in this case, but there is a clever workaround known as *backpropagation.* This approach takes into consideration the layered structure of the model by computing the output error and using it to recursively compute the errors at each hidden layer. The algorithm goes as follows:

1. Compute the activation of each input neuron, a^1_j
2. Perform the weightings and activations for each hidden layer in turn by using the rule

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

3. Compute the output error as $\Delta_j^l = \frac{\partial E}{\partial a_{j^l}} \sigma'(z_j^l)$ item Propagate backwards by recursively computing the hidden errors:

$$\Delta_j^l = \left(\sum_k \Delta_k l + 1 w_{kj}^{l+1} \right) \sigma'(s_j^l)$$

4. Finally determine the cost function gradient with respect to the weights(and biases) as

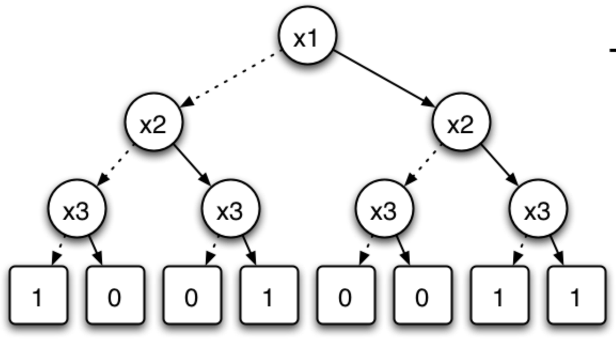
$$\frac{\partial E}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1}$$

$$\frac{\partial E}{\partial b_j^l} = \Delta_j^l$$

Random forests

Decision trees

A *decision tree* is, in its full generality, a type of directed acyclic graph where one starts at a "root" node and is presented with a question that has two or more possible answers, represented by edges flowing out of the root note and into a "leaf" node, where the process repeats itself. Eventually one ends up with a tree like structure such as the binary decision tree pictured in Fig. 4. The



x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 4: The flow starts at the node here labeled $x1$ and propagates downwards through different levels of the tree.

lowest level of a decision tree is called its "maximum depth". In the context of machine learning the data is fed through the network in such a way that each question partitions the data. These models have very high variance and are thus not suitable for classification on their own, but their relative simplicity make them ideal for use in ensemble methods. One such is the Random Forest

Results

Classification reports

```
    Classification report for classifier LogisticRegression():
              precision    recall  f1-score   support

           0       0.96      0.93      0.94        54
           1       0.96      0.98      0.97        89

    accuracy                           0.96       143
   macro avg       0.96      0.95      0.96       143
weighted avg       0.96      0.96      0.96       143

    Classification report for classifier MLPClassifier(alpha=0.2, batch_size=100, early_stop
              hidden_layer_sizes=(3, 3, 3, 3), learning_rate_init=0.01,
              max_iter=500, momentum=0.1, verbose=True):
              precision    recall  f1-score   support

           0       0.88      0.93      0.90        54
           1       0.95      0.92      0.94        89

    accuracy                           0.92       143
   macro avg       0.92      0.92      0.92       143
weighted avg       0.92      0.92      0.92       143

    Classification report for classifier RandomForestClassifier(max_depth=10, random_state=(
              precision    recall  f1-score   support

           0       0.98      0.94      0.96        54
           1       0.97      0.99      0.98        89

    accuracy                           0.97       143
   macro avg       0.97      0.97      0.97       143
weighted avg       0.97      0.97      0.97       143
```

Discussion

All models fare decently well with the parameters above. Of note is the lower precision for benignancy in the multilayer perceptron relative to the other case.

Conclusion

More fine tuning would likely have revealed how the model performances depend on the parameters.

References

Mehta et al., (2019). A high-bias, low-variance introduction to Machine Learning for physicists