**Predict Internet Popularity By Optimising Neural Networks With Python**

In the [previous post](https://www.mydatahack.com/predict-internet-popularity-by-optimising-neural-networks-with-r/), we used grid search to find the best hyper parameter for the neural network model with R’s caret package. Here, let’s use Python and scikit-learn package to optimise a neural network model.

Just like the caret package, scikit-learn has a pre-built function for hyper parameter search. As for dataset, we will use [Online News Popularity Data Set](http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity) from the UCI Machine Learning repository, which is the same dataset used in the [previous post](https://www.mydatahack.com/predict-internet-popularity-by-optimising-neural-networks-with-r/). It was originally used in [this publication](https://repositorium.sdum.uminho.pt/bitstream/1822/39169/1/main.pdf).

The nnet package from previous post is a single hidden layer back-propagation network. Therefore, there is only one size parameter for the layer to tune. Here, we will use [MLPClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier) that implements a multi-layer perceptron algorithm. Having a multiple hidden layers enables us to specify the number of layers as well as the number of neurons for each layer.

As for parameter optimisation, we will use [GridSearchCV](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) with 10-fold cross validation. Optimising multi-layer neural networks can be lengthy because you can try different layer numbers and neuron size in each layer. Here, let’s use 2 layers and try to optimise the neuron size per layer for simplicity. Generally speaking, stating small works fine with neural networks in terms of both layer and neuron sizes.

We will also search for the best alpha parameter for L2 regularisation (you can read more about regularisation [here](http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/)). In short, regularisation prevents over-fitting by penalising it. With no regularisation term, you will get a great accuracy on training set, but not so great on test. Neural networks are often called as a black box method, which makes the method fancy and magical. In realisty, they are a variation of nonlinear statistical models. So, regularisation terms work just like logistic regression. Scikit-learn has a great documentation on MPL [here](http://scikit-learn.org/stable/modules/neural_networks_supervised.html) for more detail.

As in the previous post, we are dealing with the prediction of binary classifier (‘popular’ or ‘unpopular’) based on the attributes comes with online news paper articles (see details [here](https://www.mydatahack.com/predict-internet-popularity-by-optimising-neural-networks-with-r/)).

OK, let’s code!

**Summary Steps**

* Get data and prep it (by selecting the right columns, splitting them to training and test and normalising the data).
* Do hyperparameter search.
* Model & predict with the best hyper parameter.
* Calculate performance metrics and draw ROC curve.

**Code**

1. First of all, we have to load the data (after downloading data from [here](http://archive.ics.uci.edu/ml/datasets/Online+News+Popularity)) and do data prep. I normalised it after splitting into training and test set, but you can do it prior to the split. Note that you have to use the scaler function fitted with training data for the test data set to keep consistency if you are doing this way.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

1. I created a grid\_search function so that I can reuse it for other models. It takes a scikit-learn model, feature matrix, label array and parameter grids and return the best parameters.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

1. See the range of the alpha and neuron size for each layer that I choose. For some reason, neuron size (5, 2) appear in many places (including the official documentation). So, I based my parameter grid choice on them and changed slightly. The bigger the neuron size is, the more complex the models become. Generally speaking, a smaller neuron size works for most of the time. But, of course, you have to experiment it a lot to find the best size. You can read [this post](http://www.heatonresearch.com/2017/06/01/hidden-layers.html) for more details. It is a good starting point to know about layer and neuron size.

I picked lbfgs for a solver. You can try to use different ones. Again, you have to experiment it to see which one works best for your use case.

As I did with R code [here](https://www.mydatahack.com/predict-internet-popularity-by-optimising-neural-networks-with-r/), you can do parallel computing by adding the n\_jobs=-1. As the grid search is an iterative CPU intense process, it is always faster to do parallel.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

Best F1 Score is: 0.6631095339771439

Best Parameter is: {'alpha': 3e-05, 'hidden\_layer\_sizes': (5, 2)}

1. Now, we got the best hyperparameter set, let’s model the neural net and do prediction.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

1. I created two functions to get all the evaluation metrics. The calculate\_auc function also produces ROC. I used pandas magic to create a data frame for the easy summary of performance metrics. These metrics are the same one use in the [publication](https://repositorium.sdum.uminho.pt/bitstream/1822/39169/1/main.pdf).

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

Here is the output.

**Comparing it With Default MLP Classifier Model**

If you don’t set any parameter, the default alpha is 0.0001 and hidden\_layer\_sizes is 100 neurons in a single layer. The default model is over-fitting, which happens a lot for neural networks. You can see the power of simple parameter tuning!

[cc lang="python" tab\_size="4" lines="-1"]

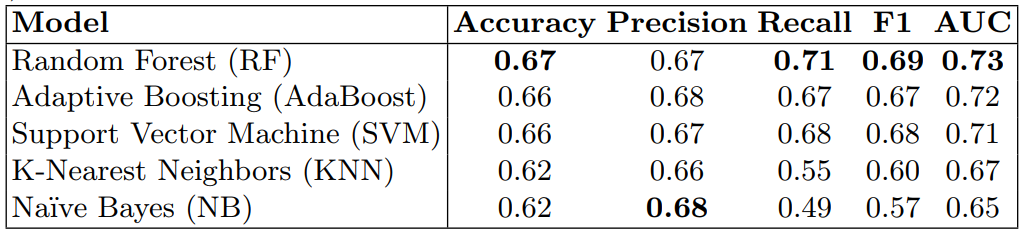
[/cc]

Neural Network Model Train Accuracy: 0.77

Neural Network Model Test Accuracy: 0.62

**Let’s Benchmark!**

Below is the performance metric table from the paper (Fernandes et al. 2015). Our neural net model was comparable to the second best method (AdaBoost) in terms of accuracy and AUC. The model is still over-fitting slightly and has lower recall. So, I think there are more room for optimisation. All in all, pretty good outcome considering how simple the whole modelling process was.



**Your Turn**

In the [publication](https://repositorium.sdum.uminho.pt/bitstream/1822/39169/1/main.pdf), they also used grid search to find the best hyper parameter set in a few different methods. In the paper, SCV was optimised with C ∈ {2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6} , RF and AdaBoost was with number of trees ∈ {10, 20, 50, 100, 200, 400} and KNN was number of neighbors ∈ {1, 3, 5, 10, 20}. Now that we have a custom grid\_search function, you can try finding the best hyper parameters for all these methods in a streamlined manner.

[cc lang="python" tab\_size="4" lines="-1"]

[/cc]

See how you go!

**Reference**

1. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 – Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal (https://repositorium.sdum.uminho.pt/bitstream/1822/39169/1/main.pdf)