Classification of Particle Existence Using Dense Neural Networks

Jason McDonald

Problem Statement:

The client previously inquiring about predicting superconductors has returned with a substantially larger dataset containing data about particle existence. The task asked of the study is to classify the existence using a dense neural network with a binary classification of a particle.

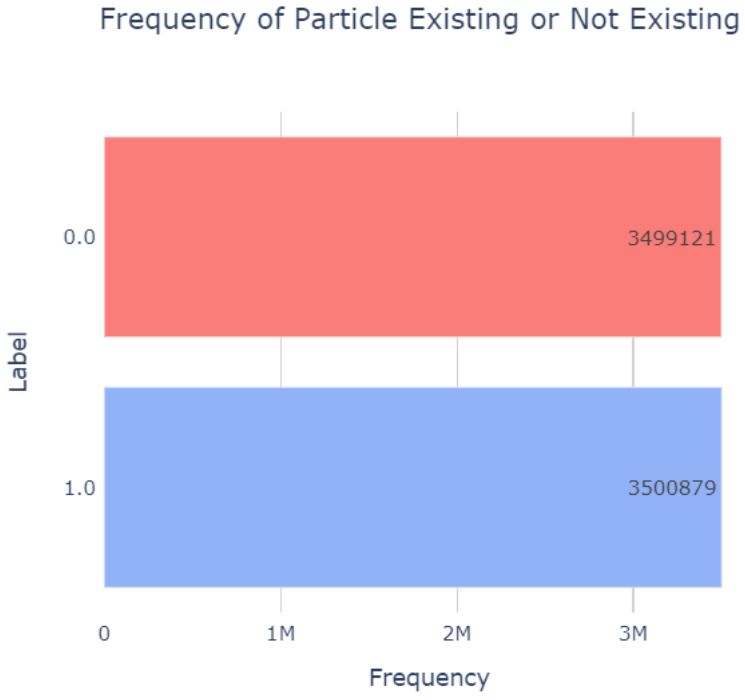
Provided Data and Evaluation:

The data provided by the client consists of 28 numeric features represented by f0 to f26 with mass as an additional numeric feature. Finally, a binary label was provided as identification of whether a particle exists (=1) or does not exist (=0)

Quantity of Data: The dataset contains 7 million rows of labeled data.

Missing Data: The data provided does not contain any missing values.

Response Variable: The response variable is a binary variable used to describe the existence of a particle using either a 1 for a particle exists or 0 if it does not exist.



The dataset does not exhibit any worrisome imbalance between the class with each coming in very close to 50% of the total.

Given the details of the dataset, the only preprocessing that will need to occur is to standardize the data prior to training the neural network using the StandardScaler function in the SciKit Learn Library..

Dense Neural Network for Identifying Particle Existence:

To begin the process of building a neural network, the data will be split into a feature array X and a target array y.

The feature array X will then be standardized before splitting the data into a train, test, and validation sets. The training data set will contain 70% of the data using a random selection process. The test and validation hold out will each contain 15% of the remaining rows, randomly selected for each.

A dense neural network consists of a number of layers with a final dense layer where, for classification tasks of a binary problem, like this where only two classes exist, a sigmoid activation function is used to choose which class the input variables belong within.

This study used a Hyperband tuning algorithm (Li et al., 2018) from the Keras_tuner library. The hyperband algorithm approaches hyperparameter tuning "as a pure-exploration non-stochastic infinite-armed bandit problem where a predefined resource like iterations, data samples, or features is allocated to randomly sampled configurations" (Li et al., 2018). The intent is to more quickly converge on an optimum set of hyperparameters through a round robin bracketed approach where the best set continues on to the next round in a 'winner takes all' tournament. The algorithm can and did in this case make use of an early stopping callback. The validation loss was used as the early stopping callback in this study.

The Hyperband algorithm identified a local optimum on the following hyperparameters:

Hyperparameters Identified by the Hyperband Algorithm

Hyperparameter	Value
Number of Layers	4
Layer 1 Units	384
Layer 2 Units	312
Layer 3 Units	448
Layer 4 Units	472
Learning Rate	0.0001

Table: Hyperparameters Identified by the Hyperband Algorithm lists the hyperparameters that won the bracketed style round robin tournament that the algorithm uses

The chart below demonstrates the training and validation loss and accuracy. The early stopping callback featured a patience of 5, which is why training ended and was stopped after epoch 20, when the validation loss ceased to decrease after 5 consecutive epochs.

Conclusion:

After training the final model, the model was evaluated against the test data set, which consisted of a held out 15% of the original data.

On this data, an overall accuracy of 88.54% was obtained. Each prediction is made in just over 1 millisecond.

Reference:

Li, Lisha, and Kevin Jamieson. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." Journal of Machine Learning Research 18 (2018): 1-52.