NeuralNetworkVisualization

A dynamic approach with R shiny

Alex Afanasev and Jacqueline Seufert

Advanced Statistical Programming with R

M.Sc. Applied Statistics
Georg-August-University Göttingen

September 18, 2019

Contents

1 Introduction

Interpretability is one of the key requirements for statistical analysis. In Machine Learning, the main concern is the predictability for two reasons. On the one hand, if the decision process of a model has no significant consequences for the model choice, the explanation of this decision is irrelevant. On the other hand, if the particular problem is grasped well enough in the real application, the decision of the model can be generally trusted. The interpretability of a model is desirable if the formalization of the problem is not complete. Henceforth, there exists a fundamental barrier between optimization and evaluation of the model (Doshi-Velez and Kim 2017). As the two conditions are usually not fulfilled in reality, interpretation poses a challenge in most machine learning models.

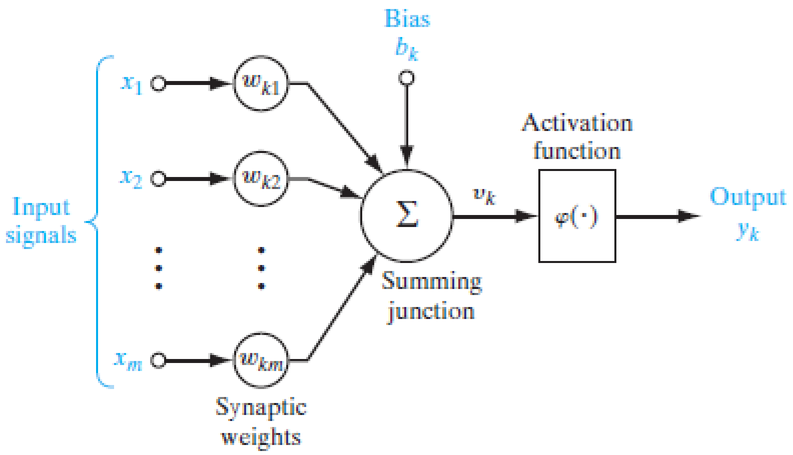
An approach from classic statistics is the interpretation of marginal effects. Neural networks are black box models, as the internal observation of their functioning is not possible. In order to determine the effect of predictors, one does not aim at opening the black box, but to observe the behavior of the black box at the change of the input. One of the methods to achieve this goal is the partial dependence plot (Friedman 2001). In order to implement this approach, we created the package `NeuralNetworkVisualization` in the software R. This paper aims to offer a dynamic approach for visualizing the partial dependence plots with R Shiny. This report is structured as follows: The first section assesses the theory which lays the foundation for the implementation of the code. The second chapter introduces and examines the created package. The next section explains the code structure. For illustration, the fourth chapter then gives examples. The last section concludes the realization of partial dependence plots.

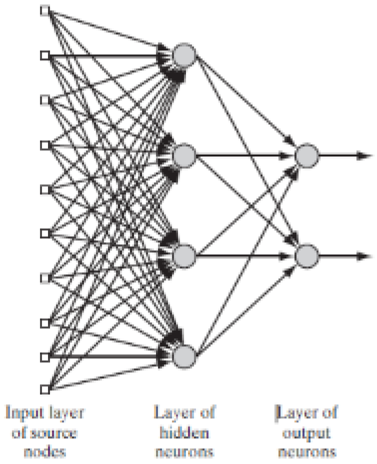
2 Theory

2.1 Neural Networks

As the goal is to visualize marginal effects of neural networks, it is imperative to gain some further insight on the functioning of these models. This section, however, aims only to give a broad overview. A detailed description of neural networks is beyond the scope of this paper. The neural network was inspired by the processes which take place in neurons in the human nervous system. A neural network is defined as a massively parallel distributed processor consisting of simple processing elements which has a natural proclivity for storing knowledge based on experience and making it available for use. It bears a resemblance to the brain in that the network obtains knowledge from its environment through a learning process and connection strengths between neurons are used to save the gained information (Haykin 2009).
Mathematically, a neural network has the following structure:

1. A state variable n_i is connected with each node i.
2. A real-valued weight w_{ik} is related with each link (ik) between two nodes i and k.
3. A real-valued bias θ_i is associated with each node i .
4. A transfer function $f_i[n_k, w_{ik}, \theta_i, (k \neq i)]$ is defined, for each node i, which determines the state of the node as a function of its bias, of the weights of its incoming links, and of the states of the nodes connected to it by these links.





The transfer function is usually of the form $f(\sum_k w_{ik} n_k - \theta_i)$, where $f(x)$ is either a discontinuous step function or a smoothly increasing generalization known as a sigmoidal function. First, each neuron adds up the value of every neuron from the previous column it is related to. Subsequently, this value is multiplied by the weight. Each connection of neurons has its own weight which is also the only value in the process that will be modified. Furthermore, a bias chosen beforehand may be added to the total value calculated. After the summation, the neuron finally applies the transfer function to the obtained value (Müller, Reinhardt, and Strickland 1995). To break down the concept of neural networks: One can observe and understand the input as well as the output layers. However, the calculations within the hidden layers remain a black box.

Neural networks have some useful properties, including: Neural networks have the ability to learn and model non-linear and complex relationships, useful for tackling real-life problems. Furthermore, they can generalize after learning from inputs, they can infer unnoticed relationships on unseen data. Thereby, the model can generalize and predict on unknown data. Lastly, these models also do not impose any restrictions on the input variables, allowing for greater flexibility. In addition, they appear to deal better with heteroskedasticity (Haykin 2009).

2.2 Marginal Effects

Marginal effects are a useful interpretation tool widely used in traditional statistics. However, as explained beforehand, machine learning methods might also require interpretability. Neural networks are part of the supervised learning environment. These models have a high-dimensional prediction function which cannot be presented in a similar algebraic fashion such as parametric model and therefore, are considered a black box (Hooker 2007). In order to tackle the opaqueness of these models, while maintaining interpretability alongside predictive power, a post-hoc analysis is most appropriate. For convolutional neural networks, there is a model-specific approach as proposed by Zeiler and Fergus (2014). However, to the best of our knowledge there exists no generic procedure for the totality of neural networks. Thus, a model-agnostic take might be more appropriate. Model-agnostic methods are applicable to every supervised learning approach and also offer the majority of available appropriate techniques. One distinguishes between global and local approaches within the model-agnostic environment. A strictly local interpretation describes the prediction of single data points. Global techniques interpret model functionality for the entire dataset. Local and global approaches are linked to each other by disaggregation (Molnar 2018).

2.3 Partial Dependence Plots

The global, post-hoc approach that we will apply is the partial dependence plot. The concept of this plot was first introduced by Friedman (2001). The approach is very straightforward: Let z_l be a chosen subset of interest, of size l, of the predictor variables x, of size n,

$$z_l = \{z_1, ..., z_l\} \subset \{x_1, ..., x_n\}$$

and $z_{\setminus l}$ be the complement subset, such that $z_{\setminus l} \cup z_l = x$. The prediction $\hat{F}(x)$ depends on variables in both subsets: $\hat{F}(z_l, z_{\setminus l})$. In general, the functional form of $\hat{F}_{z_{\setminus l}}(z_l)$ will depend on the particular value chosen for $z_{\setminus l}$. If this dependence is not too strong, then the average function

$$\bar{F}_l(z_l) = E_{z_{\setminus l}}[\hat{F}(x)] = \int \hat{F}(z_l, z_{\setminus l}) p_{\setminus l}(z_{\setminus l}) dz_{\setminus l}$$

can represent the partial dependence of $\hat{F}(x)$ on the chosen predictor subset z_l. Here $p_{\setminus l}(z_{\setminus l})$ is the marginal probability of $z_{\setminus l}$, $p_{\setminus l}(z_{\setminus l}) = \int p(x) dz_l$, where $p(x)$ is the joint density of all of the inputs x. This complement marginal density can be estimated from the training data, so that the previous equation becomes

$$\bar{F}_l(z_l) = \frac{1}{N} \sum_{i=1}^{N} \hat{F}(z_l, z_{i,\setminus l})$$

To illustrate, let $z_s = x_1$ be the predictor subset of interest with unique values $\{x_{11}, x_{12}, ..., x_{1k}\}$. The partial dependence of the response on x_1 can be created in the following steps: For $i \in 1, 2, ..., k$

1. Duplicate the training data and replace the original values of the predictor of interest x_1 with the constant x_{1i}
2. Compute the vector of predicted values from the transformed copy of the training data.
3. Calculate the mean prediction to obtain $\bar{f}_1(x_{1i})$

Plot the pairs $x_{1i}, \bar{f}_1(x_{1i})$ for $i = 1, 2, ..., k$. (Greenwell 2017)

2.4 Discussion of Partial Dependence Plots

The calculation of partial dependence plots is intuitive: The partial dependence function at a specific predictor value represents the average prediction if we force every data point to take that particular value. In the case where the feature of interest is uncorrelated with other predictors, the interpretation is evident: The plot displays how the average prediction changes when the j-th predictor of interest is changed. Moreover, the method is easy to implement. The calculation for the plots has a causal interpretation. We modify a feature and quantify the changes in the prediction. However, there are also some downsides. Partial dependence plots can plot only up to three features at once due to the constraints in visualizing multidimensionality. Yet, we refrain from this approach and simply plot one-dimensional relationships.

Some plots do not show the predictor's distribution. Omitting the distribution can be obfuscating, because one might overinterpret regions with little data. The main problem is the assumption of independence between features.

When the features are correlated, we create new data points in areas of the feature distribution where the actual probability is very low (Molnar 2018). One solution to this problem is Accumulated Local Effect plots which integrate over the conditional instead of the marginal distribution (Apley and Zhu 2016).

Furthermore, hidden heterogeneity might be an issue because partial dependence plots only show the average marginal effects. By plotting the individual conditional expectation curves rather than the aggregated line, we can reveal heterogeneity (Goldstein et al. 2013). The in-depth discussion of these methods falls outside the scope of this paper.

3 Introducing the `NeuralNetworkVisualization`-package

We created a package for R which produces partial dependence plots in different presentations and for different types of data. The technical details of this package will be discussed in the next section.

There exist two other packages which also implement partial dependence plots. One of them is the `pdp`-package (Greenwell 2017). The algorithm is the same among all the packages, but they do vary in visualization and options to customize the plots.

In the `pdp`-package, the graphics are based on `lattice`. Moreover, it is possible to plot multiple features at once. One can plot two-dimensional plots alongside three-dimensional surfaces. In addition, it offers solutions for constructing plots in regions outside the area of the training data. The `pdp`-package also allows for constructing the plots in parallel. Furthermore, the package allows for any object of supervised learning ranging from random forests over generalized linear models to neural networks.

The other package which allows plotting partial dependence plots is `plotmo` (Milborrow 2019). By default, `plotmo` sets every predictor to their median apart from the subset of interest whereas in a partial dependence plot at each plotted data point the effect of the remaining predictors is averaged. However, `plotmo` also offers multidimensional partial dependence plots. Another downside of the plotmo package is that is not very robust in the sense that it struggles to cater different requirements for different object types. Moreover, the visualization for categorical data is often imprecise. One benefit in comparison to pdp though is that `plotmo` allows the specification of a prediction interval.

Our package, `NeuralNetworkVisualization`, attempts to combine and extend the benefits of both packages. The package is limited to neural networks and is restricted to one-dimensional plots. It offers the user to customize the neural network and allows for bootstrap intervals sped up by parallel computing. The user can choose between a visualization with `ggplot` or `plotly`, which offers additional, interactive information on the plot. It also offers options to plot multiple feature plots at once for any data type. In addition, `NeuralNetworkVisualization` can create dynamic plots on R shiny allowing the user to customize the visualization in an online app.

4 Code structure

Our package consists of four core functionalities:

1. Fitting the model with the `NeuralNetwork` class
2. Preparing data for visualization using the `prepare_data` function
3. Creating partial dependence plots using the function `plot_partial_dependencies`
4. Create the visualizations inside a shiny app with the `run_shiny_app` function

4.1 `NeuralNetwork`

Implementing a package that is able to create partial dependence plots requires having the ability to fit neural networks easily. `neuralnet` is one of the most common R packages used for fitting neural networks but it has some minor inconveniences. First of all, factor predictor variables have to be expanded to a set of dummy variables. In the `neuralnet` package this has to be done manually before fitting the model using the `class.ind` function from the `nnet` package. Furthermore, if the range of the predictors is different, scaling or data normalization has to be performed. Again, in the `neuralnet` function this has to be done before fitting the model. Additionally, our package functionality has as a requirement to remember all network specifications provided by the user since we need to refit the same model multiple times for creating the bootstrap confidence intervals for the partial dependence plots. Hence, we decided to write a wrapper class around the `neuralnet` function.

The `NeuralNetwork` class takes similar inputs as the `neuralnet` function and keeps track of all inputs for further model refitting for the confidence intervals. Furthermore, users don't need to expand factor variables to a set of dummy variables before fitting the model. This is being handled automatically inside the `NeuralNetwork` class. Besides that, a scale parameter has been added with which users can decide if they want the data to be normalized. Additionally, it is possible to compute the confidence interval via the bootstrap procedure before creating the visualizations by providing the `options` parameter. There the user has to specify a `list` with the parameters: `store` as a Boolean representing if the data should be stored, `parallel` as a Boolean representing if parallel computation should be used, `probs` as a vector for the lower and upper bounds and `nrepetitions` for the number of bootstrap iterations. We decided to add the parallel computation possibility for the bootstrap procedure since it decreased the running time immensely. Everything corresponding to the model fitting part has been saved in the **neural_net_class.R** file.

4.2 `prepare_data`

Data preparation for plotting is hidden from the end user. The `prepare_data` function is responsible for the plotting data creation as explained with the algorithm in chapter 2.3. Besides preparing the plotting data, the function also adds the lower and upper levels for the confidence intervals if wanted by the user. All created visualizations, if no data was stored in the fitted model via the `options` parameter, use this function to get the necessary data for plotting. Moreover, if the scale parameter of the `NeuralNetwork` has been set, the data will be descaled back to the original data range for plotting. The functionality for the data preparation is stored in the **plotting_data_preparation.R** file.

If the confidence interval should be added, keep in mind that for each bootstrap iteration the neural network will be refitted with sampled data. This procedure might take some time, since fitting a neural network depends on the available data, the number of hidden layers and the threshold for the minimization of the prediction error. As mentioned, we added the possibility to use parallel computing for decreasing the running time of the bootstrap procedure. Additionally, it is possible to store the plotting data directly in the `NeuralNetwork` class via the options parameter. This removes the waiting time for the wanted visualizations.

4.3 `plot_partial_dependencies`

After having fit the model, the user can jump directly into producing partial dependence plots with the `plot_partial_dependencies` function. There the user can select all (default behavior), multiple or just a single predictor for which to create the partial dependence plot. Besides that, one can select the lower and upper quantiles for the confidence intervals, the number of bootstrap iterations, whether it should be a `ggplot` or `plotly` visualization, if the stored data should be used and if parallel computation should be used. Of course, if the user decides to use the stored data, no computation will take place and hence changing the `probs`, `nrepetitions` and `parallel` parameter will have no effect. The **neural_net_visualization.R** file contains the plotting function definitions.

4.4 `run_shiny_app`

The partial dependence plots can be created via a shiny user interface with the `run_shiny_app` function. There the user has the possibility to upload the `NeuralNetwork` as a `.rds` file, use a model from the global environment or use one of the available example models for testing. Afterwards it is possible to decide whether to use the stored data, if available, or to calculate bootstrap confidence intervals with specified settings.

We added error messages for impossible lower and upper bound combinations, creating plots without specifying the predictors of interest and other inadequate specifications inside the shiny app, for the `plot_partial_dependencies` function and the `NeuralNetwork` class.

5 Example

This chapter will explain how to use the R package. The examples will be based on the famous iris dataset where a neural network will be used to predict the flower species based on the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris.

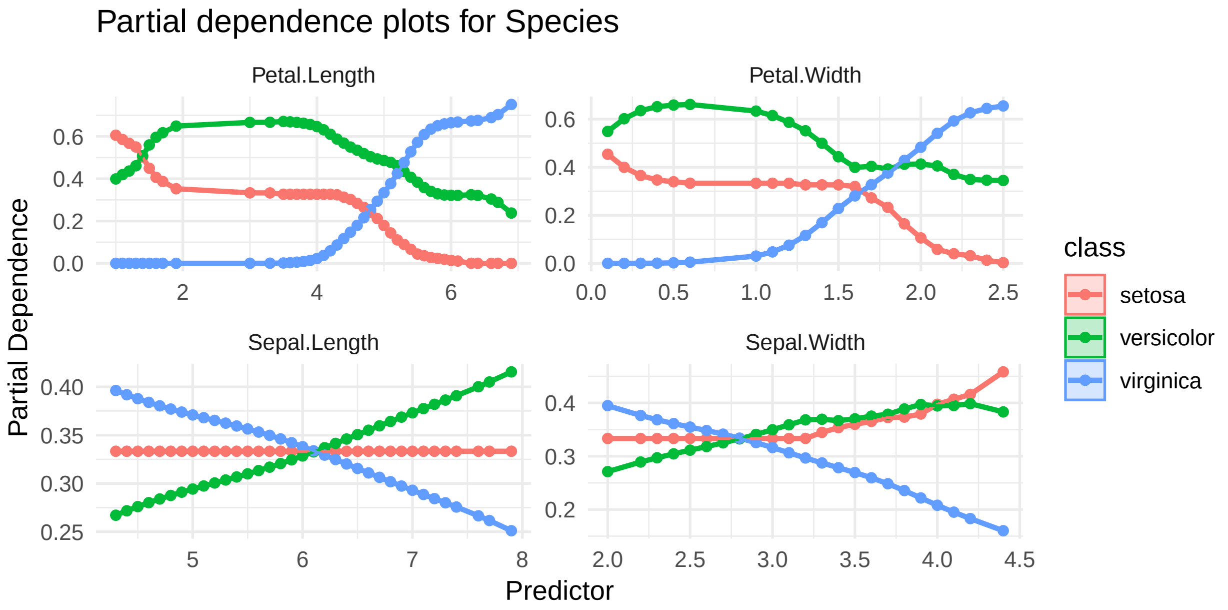
First of all, let's fit the neural network using the `NeuralNetwork` class. Compared to the `neuralnet` package less data preprocessing has to be done by the user. He or she does not have to think about factor columns that should be expanded into dummy variables and scaling does not need to be done manually. By providing the `options` parameter the data that will be used for visualizing the partial dependencies will be directly computed. The model fitting will take time, since 4000 neural networks will be refitted by the bootstrap procedure and the resulting data will be stored inside the `NeuralNetwork`.

```r
library(NeuralNetworkVisualization)
library(datasets)
model_data <- data("iris")

set.seed(1)
model <- NeuralNetwork(
    Species ~ ., data = train_model, layers = c(5, 5), rep = 5,
    linear.output = FALSE, scale = TRUE, err.fct = "ce", stepmax = 1000000,
    threshold = 0.5, options = list(
        store = TRUE, parallel = TRUE, nrepetitions = 1000,
        probs = c(0.05,0.95)))
```

Now, it is easily possible to create the partial dependence plots using the `plot_partial_dependencies` function. If the user just provides the model as input, the function will create the partial dependence plots for all predictors without a confidence interval.

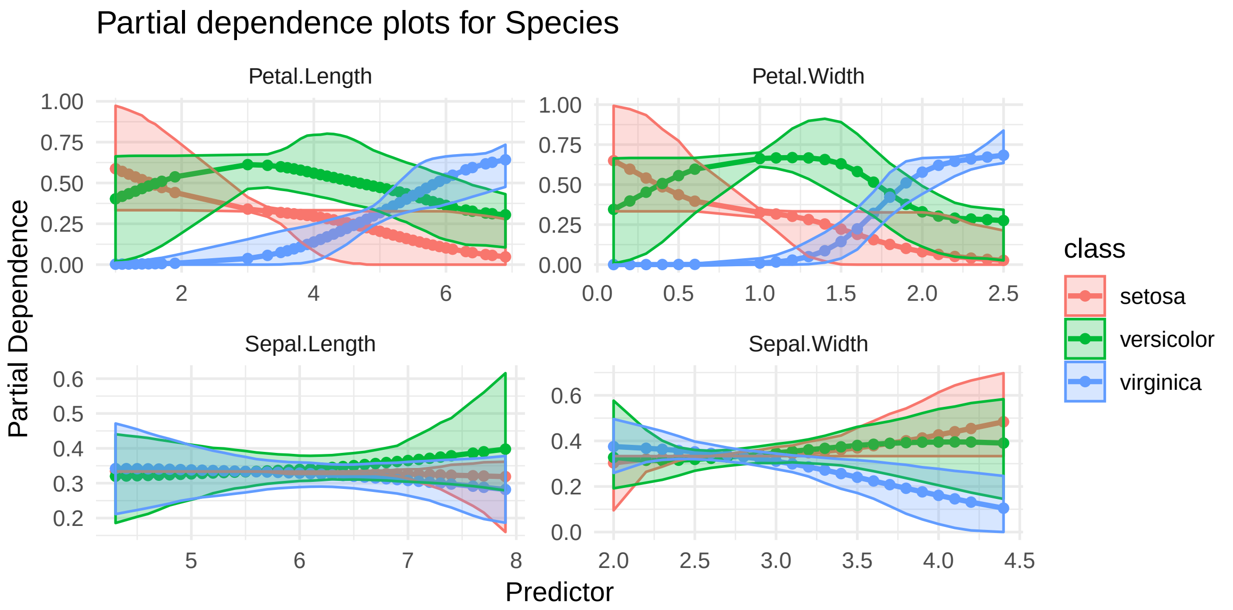
```r
plot_partial_dependencies(model)
```



The partial dependence plots show how the probability of a flower being in a specific category changes, on average, given a certain value for the predictor of interest, based on the assumption that the predictors are uncorrelated with one another.

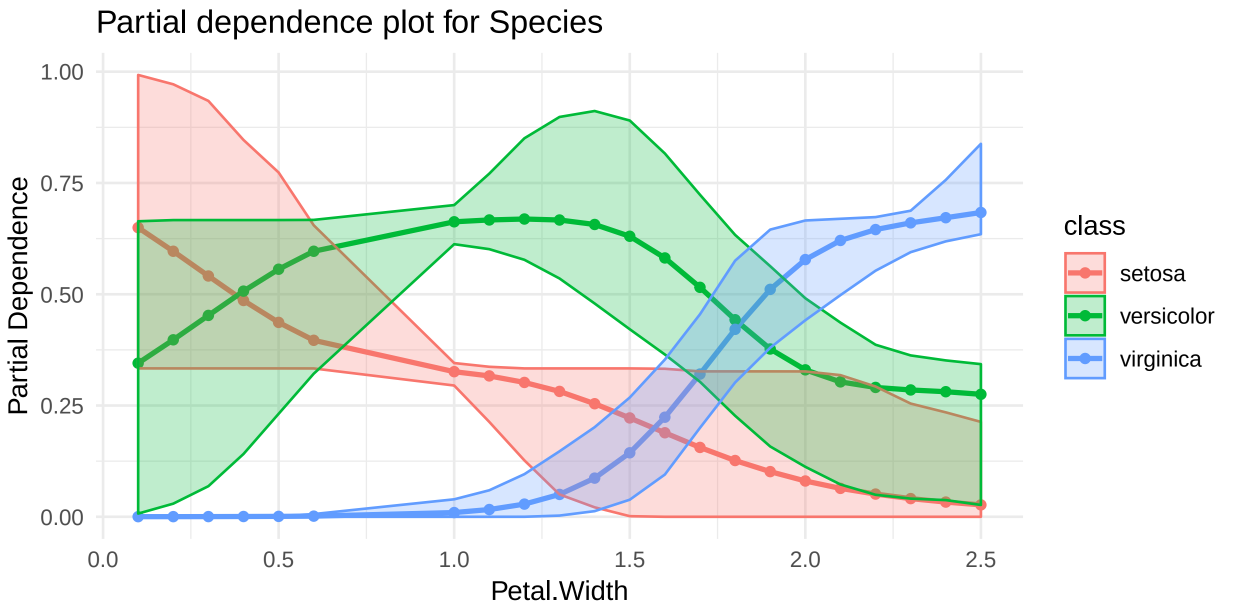
Let's take a look at the partial dependence plots with bootstrap confidence intervals to capture the uncertainty of the effect. The thick line will represent the average partial dependency and the lower and upper bound the respective quantiles. We can just use the stored data to create the partial dependence plot with the `probs` and `nrepetitions` parameter specified as in the `options` parameter where the model was fit.

```r
plot_partial_dependencies(model, use_stored_data = TRUE)
```



Now that the confidence intervals have been added to the partial dependence plots, the interpretation can additionally capture the uncertainty of the effect of a change of the predictor on the response category. Furthermore, if we want to look at one predictor only, we can just specify the `predictors` parameter. Instead of using the already stored data, we will compute the data for plotting directly.

```r
plot_partial_dependencies(model, predictors = "Petal.Width", probs = c(0.05, 0.95),
                          nrepetitions = 1000, parallel = TRUE)
```



6 Conclusion

This paper gave a brief introduction into the concept of interpretability for neural network models. Especially partial dependence plots have been examined and implemented in our `NeuralNetworkVisualization` package.

Of course, there are some issues using this approach as has been shown in chapter 2.4. Hence, we do not claim that our package is a complete solution for this task. There are a number of steps that could be taken for improvement.

Firstly, considering the fact that we are only showing one-dimensional relationships, it would be possible to add visualizations for up to three predictors combined. Additionally, Accumulated Local Effect plots could also be added for further interpretability possibilities. Moreover, Individual Conditional Expectation plots are useful for detecting hidden heterogeneity that cannot be seen with partial dependence plots.

However, we created a package that has easy to use functions with which the end user can create visualizations interactively in a shiny app or with the R console. A lot of options are added, from deciding which type of plot should be created to whether or not the computations should be done in parallel. The package functionalities where tested extensively and we tried to minimize the possibilities for inadequate function arguments by adding clear error messages.

Bibliography

Apley, Daniel W., and Jingyu Zhu. 2016. "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models." https://arxiv.org/pdf/1612.08468.

Doshi-Velez, Finale, and Been Kim. 2017. "Towards a Rigorous Science of Interpretable Machine Learning." *arXiv Preprint arXiv:1702.08608.*

Friedman, Jerome H. 2001. "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics*, 1189–1232.

Goldstein, Alex, Adam Kapelner, Justin Bleich, and Emil Pitkin. 2013. "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." https://arxiv.org/pdf/1309.6392.

Greenwell, BrandonM. 2017. "Pdp: An R Package for Constructing Partial Dependence Plots." *The R Journal* 9 (1): 421. https://doi.org/10.32614/RJ-2017-016.

Haykin, Simon S. 2009. *Neural Networks and Learning Machines.* 3rd ed. New York: Prentice Hall.

Hooker, Giles. 2007. "Generalized Functional Anova Diagnostics for High-Dimensional Functions of Dependent Variables." *Journal of Computational and Graphical Statistics* 16 (3): 709–32.

Milborrow, Stephen. 2019. "Plotting Regression Surfaces with Plotmo."

Molnar, Christoph. 2018. "Interpretable Machine Learning." *A Guide for Making Black Box Models Explainable* 7.

Müller, Berndt, J. Reinhardt, and M. T. Strickland. 1995. *Neural Networks: An Introduction / B. Müller, J. Reinhardt, M.t. Strickland.* 2nd updated and corr. ed. Physics of Neural Networks. Berlin; London: Springer.

Zeiler, Matthew D., and Rob Fergus, eds. 2014. *Visualizing and Understanding Convolutional Networks.* Springer.