

REPORT

„Application of NN to calibrated rangefinder approximation”

AUTHORS:

Agnieszka Góral (agnigor672@student.polsl.pl)

Damian Wieczorek (damiwie617@student.polsl.pl)

CONTENTS:

1. Short introduction presenting the project topic	2
2. Analysis of the task	2
3. Internal and external specification of the software solution	5
4. Experiments	8
5. Summary, overall conclusions, possible improvements, future work etc.	11
6. References – list of sources used during the work on the project	12
7. Link to the Git where all the files are placed	12

1. Short introduction presenting the project topic

The goal was to implement the method of calibrating measurements from MiniLidarS in such a way that they are as close as possible to the results of measurements from Leica. In addition, the measurement error for the proposed method should be determined.

2. Analysis of the task

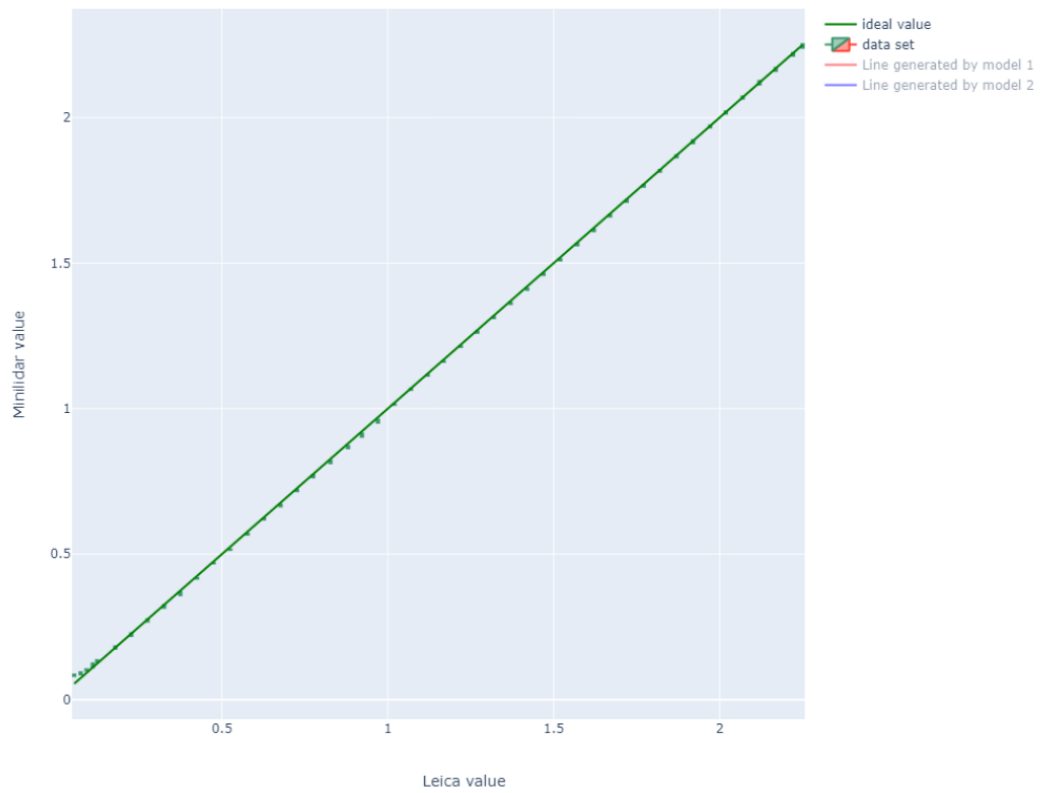
a. possible approaches to solve the problem with its pros/cons, detailed description of selected methodology:

We chose a neural network (as we had determined in topic) - is a combination of elements called artificial neurons that create at least three layers: input, hidden and output, with many hidden layers. The neural network is divided into three layers: input, hidden and output. The input and output layers depend on the format of the input and expected data. The hidden layer depends on what we want to achieve with a given network. There can be many hidden layers. The more of them, the more complex problems we can solve. A single hidden layer can solve a linear problem, but two hidden layers can solve a quadratic problem and so on. We used the Multi-layer Perceptron regressor from the sklearn library. The same model adjusts the number of neurons in the input and output layers. We created two models with different number of hidden layers to see the difference in the results. The first model has 10 hidden layers with the number of neurons from 100 to 10 decreasing by 10. The second model has only 3 hidden layers with 100, 50 and 10 neurons.

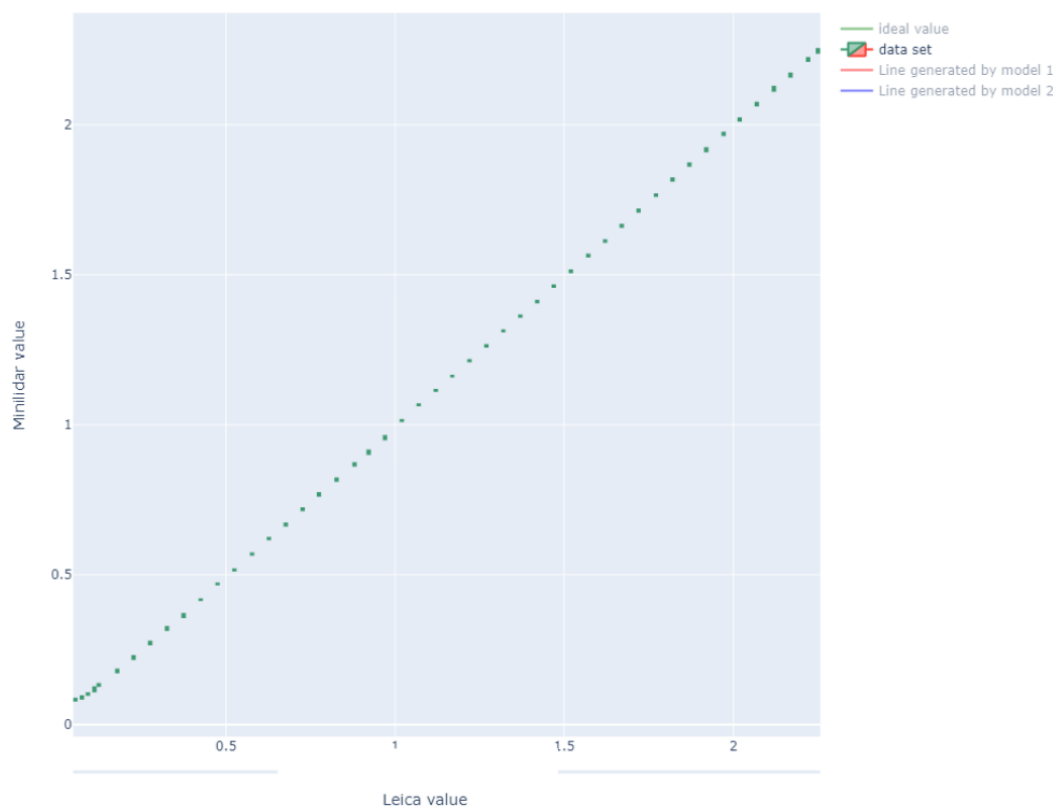
b. presentation of possible/available datasets and detailed description of the chosen ones:

We received measurements from two devices - Leica and MiniLidarS (probably laser metres). We received the data in a .csv file. In file is 1920 measurements, for each Leica value is 40 MiniLidarS values. We couldn't normalize data, because we didn't know devices used to measure. Analyzing the data set, it turned out that the MiniLidarS measurement values are overestimated for small values, and similar for larger values. We used Python to generate candle plots. Values on axes are in metres.

Plot of data



Plot of data



Finally we prepared 5 data sets to compare performances of models between different data:

- 1: default dataset without changes
- 2: 5 first values for each measurement
- 3: 5 closest values to average
- 4: dataset sorted ascending
- 5: dataset sorted descending

c. **Analysis of available tools/frameworks/libraries suitable for task solution; detailed presentation of selected tools/approach:**

We chose the **Python** programming language and the PyCharm environment. We considered using MatLab, but found it less intuitive than classic coding in the selected language.

ABOUT LIBRARIES:

- Libraries we didn't use:

Tensorflow: Tensorflow is a powerful and widely-used deep learning library. It provides a flexible architecture for building and training neural networks, including support for calibration tasks. However, it has a steeper learning curve compared to sklearn, making it less accessible for beginners. It requires more code to implement certain functionalities, which can be cumbersome for simple applications.

PyTorch: PyTorch is another popular deep learning library known for its dynamic computational graph and ease of use. It offers extensive support for neural network training and calibration tasks. However, similar to Tensorflow, it may require more effort and lines of code compared to sklearn for simpler tasks. This can be a disadvantage if simplicity and ease of use are important considerations.

- Libraries we used:

Sklearn (scikit-learn): Sklearn is a widely-used machine learning library that provides a user-friendly interface for implementing various machine learning algorithms, including neural networks. While it may not have the same level of deep learning-specific functionalities as Tensorflow or PyTorch, it offers several advantages for this task:

- Sklearn provides a consistent and intuitive API, making it easy to prototype and implement machine learning models, including neural networks.
- Sklearn integrates well with other Python libraries for data preprocessing, feature selection, and model evaluation. This enables a streamlined workflow for the entire process.

- Sklearn supports a wide range of machine learning algorithms, including regression and classification models, making it suitable for calibrated rangefinder approximation.

For the calibrated rangefinder approximation task, a neural network model was designed using Sklearn's Multi-Layer Perceptron (MLP) implementation. MLP is a type of feedforward neural network architecture that can learn non-linear relationships between inputs and outputs.

Summary of NN libraries: while libraries like Tensorflow and PyTorch are more specialized for deep learning tasks, sklearn can still be a suitable choice for the calibrated rangefinder approximation task due to its simplicity, integration capabilities, and extensibility. The Sklearn library, along with its MLP implementation and available preprocessing and calibration methods, provides a comprehensive toolkit for developing and evaluating a neural network-based solution.

Plotly: Plotly is a powerful, the most popular Python library used for creating interactive visualizations and plots. It offers a range of capabilities for generating high-quality graphs, charts, and dashboards, making it a popular choice among data scientists and analysts. We used it to generate interactive plots.

Pandas: we used to feed data to plotly. Pandas is a powerful Python library widely used for data manipulation and analysis. It provides high-performance, easy-to-use data structures and data analysis tools, making it a go-to choice for working with structured data.

Numpy: we used to feed data to sklearn model. NumPy is a fundamental Python library for numerical computing. It provides efficient data structures and functions to handle large arrays and matrices of numerical data.

3. Internal and external specification of the software solution

a) classes/objects/methods/scripts/functions etc. – depending on the approach to the project solution:

Things used from SKLEARN:

- **train_test_split**

As arguments we give x and y arrays to split those arrays in proportion of other arguments. Another argument we use is `test_size` with 0.2 value, to indicate in which proportion we want to split data. `random_state` argument we set to 42 to have control over how data is splitted and have consistent values. Other arguments we used as default values. `train_size` default value is None. `shuffle` default value is True. `stratify` default value is None.

- **MLPRegressor**

As arguments we provided value to `hidden_layer_size` which is (100,90,80,70,60,50,40,30,20,10) for first model and (100, 50, 10) for second model.
 We used `max_iter` which was set for 10000 in both models.
`random_state` was set for 42 in both models for consistent values between each run of code.
 Other arguments we used as default values.
`activation` default value is `relu`.
`solver` default value is `adam`.
`alpha` default value is 0.0001.
`batch_size` default value is `auto`.
`learning_rate` default value is `constant`.
`learning_rate_init` default value is 0.001.
`power_t` default value is 0.5.
`max_iter` default value is 200.
`shuffle` default value is `True`.
`tol` default value is `1e-4`.
`verbose` default value is `False`.
`warm_start` default value is `False`.
`momentum` default value is 0.9.
`nesterovs_momentum` default value is `True`.
`early_stopping` default value is `False`.
`validation_fraction` default value is 0.1.
`beta_1` default value is 0.9.
`beta_2` default value is 0.999.
`epsilon` default value is `1e-8`.
`n_iter_no_change` default value is 10.
`max_fun` default value is 15000.

- **fit** on MLPRegressor model

As x argument we use `xtrain` numpy array and analogically we use as y value `ytrain` variable.

- **predict** on MLPRegressor model and as argument we give `xtest` variable.

- **mean_squared_error** function to evaluate model

As first argument we provided `ytest` variable and as second we provided `y_pred` variable which is output of `predict` function from each model.
 Other values are default.
`sample_weight` default value is `None`.
`multioutput` default value is `uniform_average`.
`squared` default value is `True`.

- **score method** on MLPRegressor object also to evaluate model used

As arguments we provided `y_pred` and `xtest` variables.
 Other values are default.
`sample_weight` default value is `None`.

Things used from PLOTLY:

- **Scatter**

As arguments we provided x and y values from pandas DataFrame object

We set mode to lines.

As line argument we provided dict(color='green') value and we set name argument to "ideal value".

Another 2 Scatter plots we used had different pandas dataframe provided and other color of lines and different names.

All the other arguments were not set and default value for all of them is None.

- **Candlestick**

As arguments for this plot we provided x, open, high, low, close and open from one pandas Dataframe and we provided name of plot using name argument.

All the other arguments were not set and default value for all of them is None.

- **Figure**

As argument we provided array of plots we created earlier.

Other values are default.

layout default value is None.

frames default value is None.

skip_invalid default value is False.

- **update_layout**

As arguments we provided value to title which is equal to "Plot of data".

We used xaxis_title which we set as "Leica Value".

We also used yaxis_title which we set as "Minilidar Value".

We used height and width attribute with 1000 value.

Other values are default.

overwrite default value is False.

- **show**

This functions opens default web browser and shows plots created in code.

b) **data structures etc.:**

We used **PANDAS** library in our project for easy handling of structured data used to feed the plots functions from **PLOTLY** library.

For tsimilar reasons we used **NUMPY** library to easily transform data into multidimensional array which then was used to train, score and predict outputs from MLPRegressor model from **SKLEARN** library.

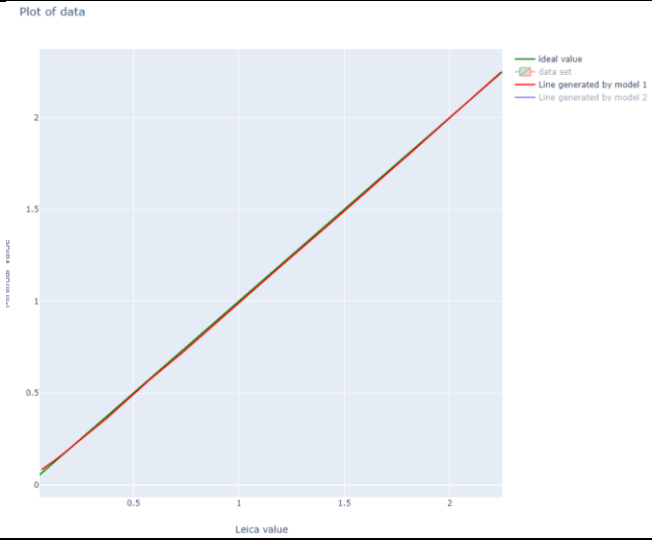
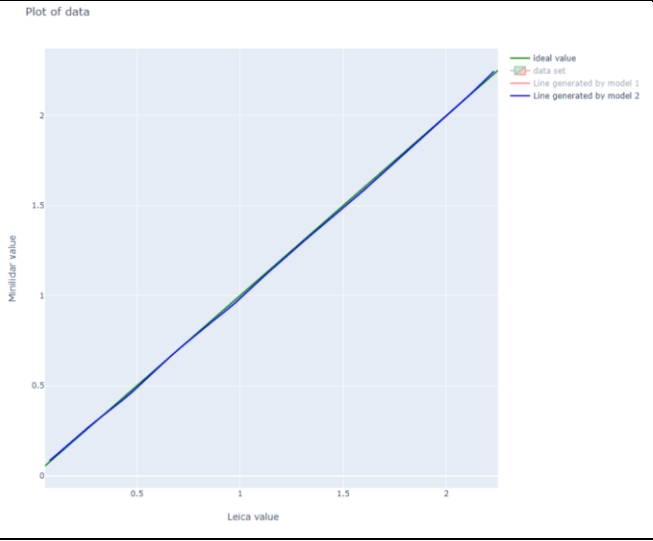
c) **user interface / GUI/console:**

Program is using console, because we didn't spend precious time on learning and creating unnecessary GUI for such a simple program. After executing the file all models get trained and all the data are printed on console and plots are showed in web browser. After that program runs in infinite loop asking user which model he wants to use or if he wants to quit program. If user choose model to use program then asks for value from MinilidarS to computer and prints it on console if value was in correct format. If value can't be casted to float value error message is shown in console. Either way program asks once again for which model user wants to choose or if he wants to quit.

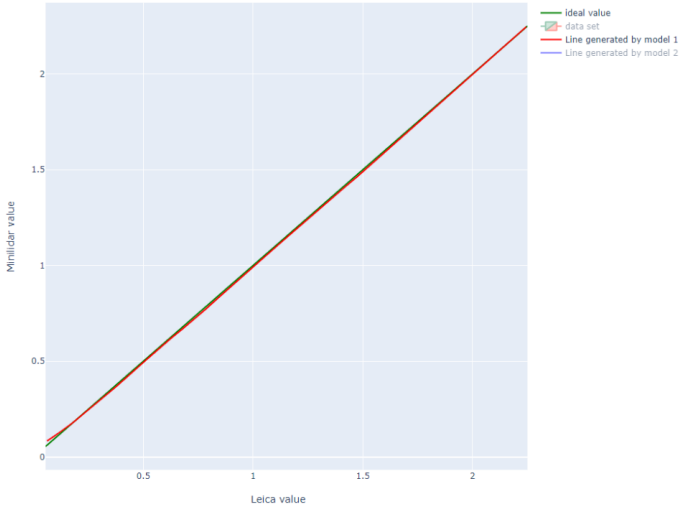
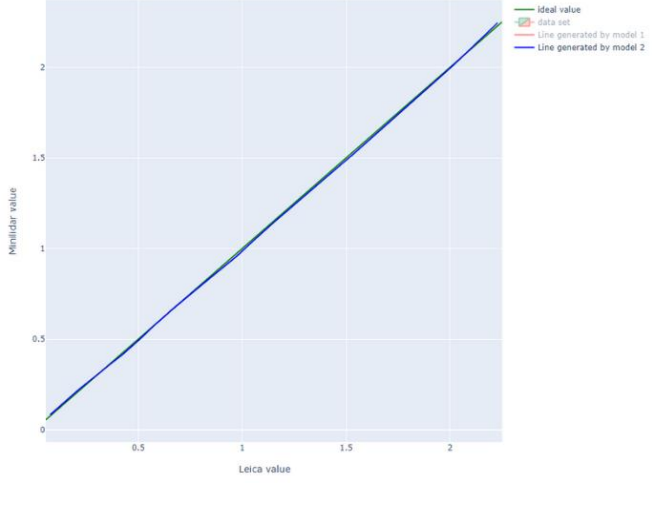
4. Experiments

- a. **experimental background** – description of experiments, what parameters/conditions were the subject of change and why, what results were observed and why; **method of results presentation** – description of diagrams/axes/values; if some raw results were processed, any calculations were performed, it must be explained (methodology, equations etc.) :

As we previous said. All datasets were divided into test and training sets with a 20/80 division. The model was exercised on a training dataset, but the mean squared error (MSE is in meters) of the model predictions was checked with a test set. The first model has 10 hidden layers with the number of neurons from 100 to 10 decreasing by 10. The second model has only 3 hidden layers with 100, 50 and 10 neurons. The table below presents the results of our neural networks depending on the training data set (plots generated by plotly library, axes – x and y values are in meters). As can be seen in the graphs above, model 2 performed worse than model 1, this is due to the smaller number of hidden layers.

DATASET	MODEL 1	MODEL 2
1		
ACCURACY	99.9368%	99.9278%
MSE	0.000011 [m]	0.00005 [m]

2	<p>Plot of data</p>	<p>Plot of data</p>
ACCURACY	98.6320%	97.2831%
MSE	0.00154 [m]	0.003971
3	<p>Plot of data</p>	<p>Plot of data</p>
ACCURACY	98.5394%	97.2937%
MSE	0.002074 [m]	0.003979 [m]
4	<p>Plot of data</p>	<p>Plot of data</p>
ACCURACY	99.94%	99.9256%
MSE	0.000011 [m]	0.000051 [m]

5	<div>Plot of data</div> 	<div>Plot of data</div> 
ACCURACY	99.9368%	99.9316%
MSE	0.000011 [m]	0.000054 [m]

b. presentation of experiments with detailed analysis and description of the results with partial conclusions possible to be drawn based on the presented part of experiments :

DEFAULT DATASET WITHOUT CHANGES			5 CLOSEST VALUES TO AVERAGE		
DATASET 1	MODEL 1	MODEL 2	DATASET 3	MODEL 1	MODEL 2
ACCURACY	99,9382%	99,9278%	ACCURACY	98,5394%	97,2937%
MSE	0,000014	0,00005	MSE	0,002074	0,003979

5 FIRST VALUES FOR EACH MEASUREMENT			DATASET SORTED ASCENDING		
DATASET 2	MODEL 1	MODEL 2	DATASET 4	MODEL 1	MODEL 2
ACCURACY	98,632%	97,2831%	ACCURACY	99,94%	99,9256%
MSE	0,00154	0,003971	MSE	0,000011	0,000051

DATASET SORTED DESCENDING		
DATASET 5	MODEL 1	MODEL 2
ACCURACY	99,9368%	99,9316%
MSE	0.000011	0,000054

MSE for both models fluctuated around zero, differences could be seen only after extending the results to 6 significant figures. The accuracy of the obtained results ranges from 98.5394% to 99.94%, which is a very good result. Both models performed best for the full dataset sorted in ascending order, and worst for the minor dataset. Thanks to the greater number of hidden layers, model 1 does better in each case, but it should be noted that these are not very large differences. Both models meet the assumptions given by the teacher. However, if we had to choose only one, it would definitely be the first model. Here we presented some results of using program:

LEICA REFERENCE VALUE [m]	0,055	0,227	0,525	0,827	1,12	1,42	1,72	2,019
MINI LIDAR S VALUE [m]	0,081	0,219	0,515	0,813	1,113	1,408	1,711	2,014
RESULT VALUE [m]	0,062	0,221	0,52	0,824	1,121	1,415	1,719	1,016

5. Summary, overall conclusions, possible improvements, future work etc.

The second model fared much worse, due to the smaller number of hidden layers. Model 1 obtained the highest accuracy for all received data sorted in ascending order. Model 1 performed better for data sorted in ascending order than for data sorted in descending order. To make the differences between some datasets visible, we had to increase the number of significant figures. We consider the task to be done correctly, as evidenced by the results achieved, we have achieved almost 100% effectiveness for data normalization. The model that can be implemented as an official solution can be model number 1 learned on the ascending order of all received data.

6. References – list of sources used during the work on the project

- I. <https://www.python.org>
- II. <https://www.ibm.com/topics/neural-networks>
- III. https://en.wikipedia.org/wiki/Artificial_neural_network
- IV. <https://www.techtarget.com/searchenterpriseai/definition/neural-network>
- V. <https://scikitlearn.org>
- VI. <https://plotly.com/python/>

7. Link to the Git where all the files are placed

<https://github.com/agnieszkag1/Application-of-NN-to-calibrated-rangefinder-approximation>