Analyzing large Datasets and predicting Outcomes using State of the Art Data Science Methods by taking the Example of the Mercedes-Benz Greener Manufacturing Campaign

Preprint · July 2021

DOI: 10.13140/RG.2.2.29958.04162

CITATIONS
0

READS
11

2 authors, including:

Ulrich Göhner
Hochschule für angewandte Wissenschaften Kempten
72 PUBLICATIONS **45** CITATIONS

SEE PROFILE

Analyzing large Datasets and predicting Outcomes using State of the Art Data Science Methods by taking the Example of the Mercedes-Benz Greener Manufacturing Campaign

Colin Theimer

July 2021

Hochschule Kempten
University of Applied Sciences

Fakultät Informatik

Analyzing large Datasets and predicting Outcomes using State of the Art Data Science Methods by taking the Example of the Mercedes-Benz Greener Manufacturing Campaign

Colin Theimer

***Abstract* – This paper is detailing a basic approach to solve Data Science Problems, going over the stages of analyzing the data, preparing it and finally solving the task using algorithms ranging from basic linear regression models up to complex neural networks in the likes of Keras.**

***Index Terms* – Data Analysis, Data Science, Outcome Prediction, Neural Network**

I. INTRODUCTION

As part of the semester project "Data Science in the industrial production" the aim was to choose from three possible projects, one being the Mercedes-Benz Greener Manufacturing Initiative hosted on kaggle. The goal of the competition was to create an algorithm that takes existing data into account to predict the outcome when new data is fed to it. The data Daimler is providing comes from a series of tests, in which they try as many possible feature combinations as possible to ensure the best safety and quality regarding their cars. However, this takes a long time and optimizing the process to shorten said time would lead to faster tests as well as reduced carbon dioxide emissions without impacting the quality of the product [1].

Analyzing a problem like the one given by Daimler is a fairly common task nowadays as more and more companies try to optimize their work-flow to spend less resources on problems that can be significantly sped up or maybe even fully automated by implementing specifically tailored algorithms.

For that reason this paper is providing a basic approach to solve data science problems like the one described above, as well as giving a fundamental starting point for strategies involving data science tasks in general.

II. DATA ANALYZATION

The first step to solve a data science task is analyzing the data. This phase can take many forms, depending on the data present. Commonly one will be faced with one or more training files, and, depending on the situation, also one or more test files. The training file(s) contain the raw data in addition to the existing information on the property you are trying to predict.

Looking at the Mercedes-Benz files, there is one training file as well as one test file. The training file contains 4209 rows and 378 columns, split into ID, y (time spent on test) besides eight columns with categorical features and 368 with numerical ones (specifically binary values). The number of rows in the test file is identical, with the columns of the test file also remaining the same minus the y-field. Each of the numerical and categorical columns represents "an anonymized set of variables, each representing a custom feature in a Mercedes car." [1] The ground truth, what is known in the context of data science as the measured result for a target variable, is represented by the y-variable in this dataset.

No matter what kind of goal you are trying to achieve, it makes sense to get a better understanding of the data by creating some plots. At this point a first decision should be made about which programming language is going to be used. Two of the most commonly used languages are Python and the R language. With R being a leading language in regards to analytics and data modeling, and Python also being most commonly used for data analytics as well as its broad spectrum of libraries [2], both are good choices for the Mercedes-Benz Initiative.

Taking a look at the ground truth's spread it becomes apparent that the Mercedes-Benz data doesn't follow the bell curve one can expect, but instead has multiple local maxima at around 77s, 89s and 112s. This means that some linear models will most likely not perform as well as others compared to usual problems of the sort.

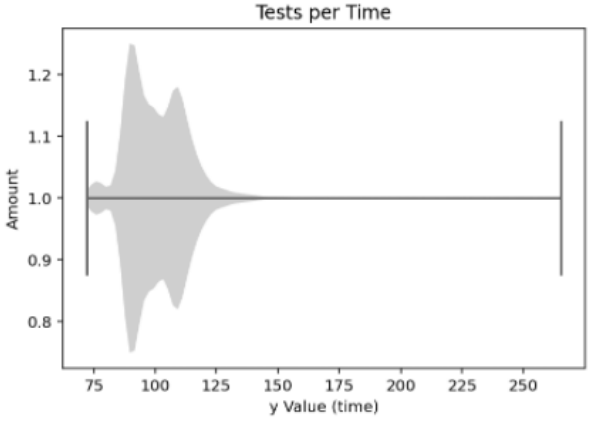

Figure 1: Spread of the y value

III. DATA PREPARATION

Data preparation is the process of cleaning and preparing the data [3]. There are a few steps that should almost always be done when handling data. If not already present, an index field should be added to the dataset. This way, each row can be monitored individually. After this, categorical features need to be turned into numeric ones. Many algorithms do not work when the dataset includes categorical columns, for example the StandardScaler present in the sklearn library or the fitting algorithms for most models. Depending on the data it may or may not make sense to standardize the numeric fields. Standardization in the context of data science refers to the process of applying scales for better comparison between columns featuring large differences in range or value (e.g. col1 := { 0; 10 }, col2 := { 0; 2000 }), which may have been caused by different measurement units [4]. Generally speaking, the larger a value is, the more influence it can have on your model. An example for this is found in the house price prediction competition [5]. Considering the column 'number of fireplaces' (range 0-3) compared to the column 'year sold' (range 2006-2010), algorithms such as the linear regression model will put a much greater weight on the year, despite the number of fireplaces having an arguably higher influence on the price of a house. However, some models are not sensitive to ranges in variables, popular examples being 'Random Forest' or the Decision Tree [4]. For that reason it makes sense not to standardize your data in these cases. Finally one should figure out a method to deal with outliers. An outlier is a data point that is outside of the expected spectrum and can be caused by "measurement errors, human errors, or data collection errors" [6] and usually lies outside three standard deviations from the mean (~0.26% of data points are outliers) [6]. Depending on the goal one is trying to achieve, outliers can be dealt with in three ways: by taking them out of the dataset, or by putting less or more weight on them. Removing outliers usually results in better general predictions, but highlighting them can result in better results for edge cases. Outliers can be manually removed from the dataset, e.g. calculating the mean of a variable and then excluding everything that isn't within three standard deviations, though algorithms like the commonly used 'Isolation Forest' from the Python library scikit, which finds anomalies by looking at "the minority consisting of fewer instances" [7] as well as the "attribute-values that are very different from those of normal instances" [7] are likely to achieve better results.

Applying those techniques to the Mercedes-Benz dataset looks as follows: The data is already linked to an index column and even though the numbers aren't continuous, there is no need to add a secondary index. As previously established the dataset contains eight columns with categorical data. Getting these can be done in a way such as this:

```
cat_cols =
df_data.select_dtypes(include=['object']).columns
```

Using one-hot-encoding, a technique that transforms a categorical feature to a vector in which there is a single 1 value and the rest being zeros [8], the categorical columns are transformed to numeric ones:

```
cat_num_cols = pd.get_dummies(df_data[cat_cols])
```

Now the old columns need to be dropped and replaced by the new ones. To scale the data, first it makes sense to remove the ID column as well as the y column from the data. After this the aforementioned StandardScaler by sklearn was used:

```
# Getting the Scaler
scaler = StandardScaler()

# Scale the Data
df_scaled =
pd.DataFrame(scaler.fit_transform(df_data))
```

This scaler standardizes "features by removing the mean and scaling [them] to unit variance" [9]. At this point the two previously removed columns can be rejoined with the scaled data. Finally the outliers are to be dealt with.

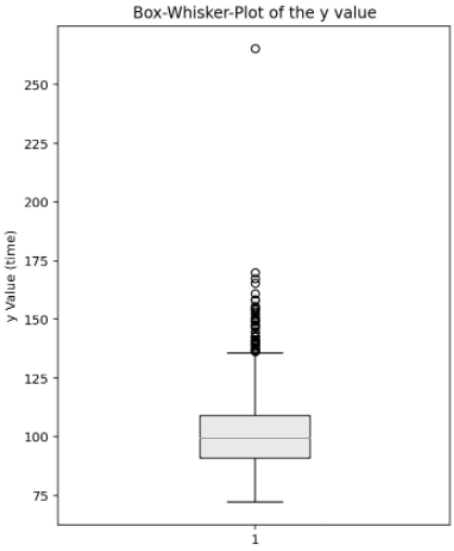

Figure 2: Outlier detection

A box-whisker plot of the y values reveals that the outliers have y-values starting at around 135s with most of them being below 150s. Therefore two models will be trained with rows featuring y values of above 135s and 150s being removed from the dataset to see which model results in better overall scores.

IV. MODEL BUILDING AND IMPLEMENTATION

This paper will specifically look at linear regression as a basic way to approach data science problems as well as the Keras model to showcase an option using a neural network.

Linear regression is a model which attempts to find relationships between two continuous variables, the independent variable (predictor) and the dependent one

(response) [10]. Hereby only statistical relationships are looked for [10], e.g. the relation between calorie intake and weight gain. While calorie intake certainly plays a part in weight gain, other factors which might not have been recorded or aren't as obvious are contributing as well [11]. With that in mind the algorithm "obtains a line that best fits the data" [10], where the distance between the data point to the regression line is as small possible [10]. This can very easily be implemented in Python using the scikit-learn linear regression model. After having the data split in a train and a test set, and both being split further in y (ground truth) and x (columns besides y), only three lines of code are necessary:

```
# Getting the Regressor
regressor = LinearRegression()

# Training the model
regressor.fit(X_train, y_train)

# Predicting y values of the test data
df_y_predicted = regressor.predict(X_test)
```

To get a general feel as to how well the model is trained, it is advised to first try out the model on data with an existing y value as to compare the predicted values with known ones. In the case of the Mercedes-Benz dataset this would happen by using only the train file and splitting it for example using sklearn's train_test_split-function. If the model gives out promising results the algorithm can be used on the test file's dataset. Besides metrics such as the difference between predicted value and true value, the basic R^2 score (regression score function) [12], the adjusted R^2, the predicted R^2 [13] or similar scores can be used to measure performance in comparison with other models.

Regarding neural networks, or in the case of computer science artificial neural networks, they are "generalizations of mathematical models of biological nervous systems" [14]. So called artificial neurons or nodes are hereby the implementation of the brains neurons, working as the elementary processing elements [14] in the algorithms. These nodes are connected via links representing axons and interacting with each other. Each neuron takes input data, performs an operation on it and then passes it as the output (activation / node value) to another neuron [15]. The links have weights, which can and will be altered resulting in a learning model [15]. However the weights can also be set explicitly [14]. Each artificial neural networks follows the same architecture being built on input, hidden and output layers [14]. While there might be no hidden layer(s), an input and an output layer are always required. While there doesn't necessarily need to be a single node in the output layer, for problems like the Mercedes-Benz Initiative it is generally required, since a single value is to be predicted per data row. This results in architectures that look as follows:

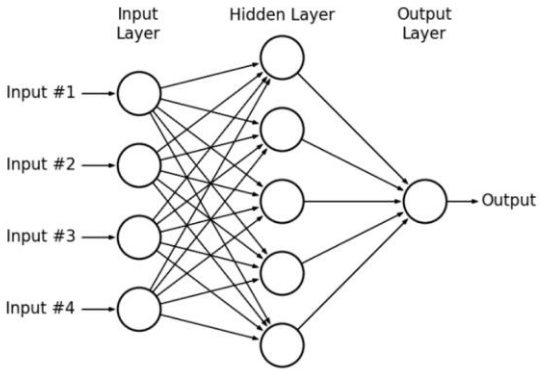

Figure 3: ANN architecture [15]

The number of nodes in the input layer depends on the amount of properties one is trying to evaluate. Say a dataset has 100 columns with one being the ground truth, the input layer would consist of 99 neurons while the output layer would be built using a single neuron. While there isn't a set amount of hidden layers or, for that matter, the amount of neurons, it is generally a good idea to start with one additional layer and half the number of neurons of the input layer. If there are two few neurons, "the network may be unable to learn the relationships amongst the data" [14]. Having too many however impacts generalizing and performance on new data [14]. Unlike models such as linear regression, neural networks can be adjusted to a great extent. Aside from the number of layers and neurons, one can also alter learning rates (updating weights during training [16]), activation functions (defining the output of a node depending on the input, e.g. ReLU, Tanh, Sigmoid, … [17]), loss functions ("quantity a model should seek to minimize during training" [18]), the optimizer (combines parameters and loss function to update the model; deeply tied in with the learning rate [19]) and others.

In turn this also means that significantly more code is needed to generate a working solution. Following will be code to solve the Mercedes-Benz Initiative:

```
# Building the layer structure
model = Sequential()
model.add(Dense(564, input_dim=564,
activation='linear'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(283, activation='tanh'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(1, activation='linear'))
# Specifying optimizer and learning rate
optimizer = optimizers.Adamax(learning_rate=0.001)
# Compiling the parameters to a model
model.compile(loss='mean_squared_error',
optimizer=optimizer)
# Training the model
```

```
model.fit(X_train, y_train, validation_split=0.33,
epochs=50, batch_size=6)

# Predicting new values

df_y_predicted = model.predict(X_test.values).ravel()
```

With that a working model has been built for both linear regression and artificial neural networks as well as the progress shown to get there.

V. CONCLUSION

	Default	Outliers	Scaled	Tweaked
Linear Regression	10.3s	8.7s	4.7s	-
Keras	11.4s	9.2s	6.7s	4.6s

Taking the best results of every step for linear regression and Keras, one can see that in the end both models perform very similar. This might be due to the fact that historically datasets with a rather small number of rows compared to the number of columns (here about 3.5:1) aren't as good a fit for neural networks as sets with a larger number of columns. Other factors might contribute as well, which shows why it is important to try out different approaches when it comes to solving data science tasks.

Despite this the model achieves an R^2-Score of about 0.61, which compares rather well with similar Keras implementations of other participants in the competition.

To further improve a model or make it more efficient during the data preparation phase one can use principal component analysis [20], Gaussian random projection [21], feature agglomeration [22] and plenty of other possibilities.

ACKNOWLEDGMENT

I would like to thank Prof. Dr. Göhner, Matthias Burkhardt and Andreas Fritz for their help supervising the project as well as Florian Zwick and Marco Wohlfart for co-developing algorithms to solve the Mercedes-Benz Initiative with me.

REFERENCES

[1] Mercedes-Benz Greener Manufacturing, kaggle.com (16.05.21): https://www.kaggle.com/c/mercedes-benz-greener-manufacturing

[2] Qamar, U. and Raza , M. S.: Data Science Concepts and Techniques with Applications, Springer Singapore, 2020, p. 11

[3] Larose, C. D. and Larose, D. T.: Data Science Using Python and R, Wiley, 2019, p. 31

[4] When and Why to Standardize Your Data?, builtin.com (25.05.21): https://builtin.com/data-science/when-and-why-standardize-your-data

[5] House Prices – Advanced Regression Techniques, kaggle.com (26.05.21): https://www.kaggle.com/c/house-prices-advanced-regression-techniques

[6] Kotu, V. and Deshpande, B.: Data Science (Concepts and Practice), Elsevier Science, 2018, p. 448

[7] F. T. Liu, K. M. Ting and Z. Zhou: Isolation Forest, Eighth IEEE International Conference on Data Mining, 2008, pp. 413-422

[8] Zheng, A. and Casari, A.: Feature Engineering for Machine Learning (Principles and Techniques for Data Scientists), O'Reilly Media, 2018, c. 5

[9] StandardScaler, scikit-learn (28.05.21): https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[10] Linear Regression – Detailed View, towardsdatascience (03.06.21): https://towardsdatascience.com/linear-regression-detailed-view-ea73175f6e86

[11] Statistical Relationship: Definition, Examples, Statistics How To (03.06.21): https://www.statisticshowto.com/statistical-relationship/

[12] R^2 score, scikit-learn (05.06.21): https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

[13] Alternatives to R-squared (with pluses and minuses), Data Science Central (05.06.21): https://www.datasciencecentral.com/profiles/blogs/alternatives-to-r-squared-with-pluses-and-minuses

[14] Abraham, A.: Artificial neural networks (Handbook of measuring system design), 2005

[15] Ali, A. and Amin, M. Z.: An Intuitive Guide of Artificial Neural Network with Practical Implementation in Keras & Tensorflow, Wavy AI Research Foundation, 2019

[16] Understand the Impact of Learning Rate on Neural Network Performance, Machine Learning Mastery (19.06.21): https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[17] 7 popular activation functions you should know in Depp Learning and how to use them with Keras and TensorFlow 2, towardsdatascience (19.06.21): https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

[18] Losses, Keras (20.06.21): https://keras.io/api/losses/

[19] Introduction to optimizers, Algorithmia (20.06.21): https://algorithmia.com/blog/introduction-to-optimizers

[20] PCA, scikit-learn (27.06.21): https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[21] Random Projection, scikit-learn (27.06.21): https://scikit-learn.org/stable/modules/random_projection.html

[22] Feature Agglomeration, scikit-learn (27.06.21): https://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html