Comparing Random Forest and XGBoost on synthetical Time Series

Jonathan K

Abstract

This paper compares Random Forest and XGBoost first theoretically and afterwards practically on Time Series data. The Time Series data involved is a synthetical generated Time Series dataset which is meant to mimic real world datasets of calls in a callcenter which shall be predicted. The variable examined is the increasing noise in the data and how the RMSE of both algorithms changes. Random Forest has always a slightly lower RMSE and seems to be more stable with increasing noise compared to XGBoost. This could be because of the bagging and bootstrapping approach of Random Forest. Yet the experiment was only conducted on one synthetical Time Series dataset. To verify the claims the result has to be repeated on more data, potentially real data, with looking into more parameters and with optimization of the hyperparameters.

Contents

List of Figures

1 Introduction

An important part of business in the Deutsche Telekom AG is Time Series prediction. As a telecommunication company it is crucial to help customers who have problems with the products or need assistance. The issue is that customer calls are not equally distributed over the week and year. They follow certain patterns and are influenced by extrinsic factors too. So the ability to predict how many complains there will be at a certain date and time is necessary to distribute employees. With proper distribution of employees it is possible to cut costs of over-staffing but also distress the workers by avoiding under-staffing.

In the field of data science and statistics there are multiple tools to predict values in a Time Series. An example of a statistical method is using the mean of customer calls that can be calculated for the last k-days to predict how many calls there will be on the next weekdays. The issue with this approach is that it is not able to utilize the relation between days and external factors, also trends apart from daily trends can not be captured by this method. Other more advanced statistical tools that will not be further discussed here are autoregressive integrated moving average (ARIMA) and the extension SARIMA. Another promising approach to predict Time Series are machine learning methods because they allow the estimation of complex functions that can capture trends, correlation and can handle data "noise". Noise describes small outliers that are always present in real world data and make it harder for algorithms to capture trends. For predicting Time Series the use of Support Vector Regression, different Neural Network architectures (LSTM ,TLNN, SANN or Transformers) and CART Algorithms are possible.[1]

This paper will only look into tree based algorithms because of the possibility to handle categorical data. Also this approach has many benefits like the possibility to track and visualize decisions. From the field of tree based algorithms two where selected (Random Forest and XGBoost) because these two are used in customer call prediction in the Deutsche Telekom AG and due to length restrictions on this paper. Random Forest and XGBoost are both algorithms used in many different domains of research due to their unique approach to prevent overfitting. [2]–[4]

This paper first compares Random Forest and XGBoost on a theoretical bases using results from other studies. Afterwards the two algorithms shall be compared on their prediction accuracy using a synthetically generated Time Series dataset with changing noise. The question to be answered is whether there are certain trends emerging in the prediction results which could help making the decision on which algorithm to use in which case easier.

2 Related Work

Tree Based Algorithms in general are widely adopted in many fields of research. They are used in sociology, medicine and many others. This is due to the possibility to track decisions and the good prediction performance when extending the default CART algorithm. [2], [3], [5] Akkaya and Çolakoğlu explored statistical and machine learning methods to predict Time Series. The conclusion was that machine learning techniques where similar or only slightly worse than the statistical approaches. Also the accuracy of the machine learning models varied strongly with the type of data given. So the usage of machine learning based methods could have benefits that are worth investigating. [2] Another paper by Krstanovic and Paulheim focused on forecasting vertical total electron content. In this paper they conclude that ensemble methods like Random Forest or XGBoost outperform regular Decision Trees especially when the prediction range is > 24h. [6]

Apart from that many papers lay more focus on researching the potential of Neural Network based models like LSTMs or Transformers. This is because of the higher availability of data and computational power that leads to Neural Networks outperforming tree based and other methods. The issue is that these high amounts of data and computational resources required for training a LSTM or Transformer are not always available so alternative methods have to be considered in many cases. [6], [7]

3 Principles of CART Algorithms

Classification and regression tree algorithms build binary decision trees out of data. They are a subset of machine learning algorithms and belong to the class of supervised learning. The CART algorithm lays the foundation for understanding the Random Forest and XGBoost algorithm so it will be explained first. There are multiple different algorithms to build decision trees. Other algorithms that will not be discussed here are the ID3 algorithm and C4.5 algorithm. [8], [9] The CART algorithm was first introduced by Breiman, Friedman, Stone, *et al.* in 1984. As the name states the algorithm can be used for both classification and regression tasks. Numerical and Categorical data can be used in these trees without the use of encoding. Also the features are not used for direct calculation so numerical features must not be normalized or standardized. Both attributes that makes data preparation easier. [10]

The CART algorithm builds the trees in a top down greedy manor. It is implemented recursively because the underlying tree data structure is recursive too. The decision which feature to prefer as a node and where to split the data is made based on a metric. The most common metric used in classification tasks is the Gini impurity. [3], [10], [11]

$$Gini(r_i) = 1 - \sum_{i=1}^{C} p_i^2 \qquad C \text{ \#Classes} \tag{1}$$

The Gini impurity or Gini index calculates the likelihood that new data is misclassified. After every split the number of positive and negative classifications can added. So p_i is the probability of the classification being positive or negative in a leaf. If y is the number of positive classifications, n the number of negative classifications and X the number of all cases in the leaf p_i for a leaf is $p_i = (\frac{y}{X}) + (\frac{n}{X})$. The Gini impurity is calculated by subtracting the sum of p_i squared from one. [12], [13]

$$Residuals(r_i) = \sum_{i=1}^{C} (y_i - \hat{y}_i)^2 \qquad C \text{ \#data, } y_i \text{ data point, } \hat{y}_i \text{ mean,} \tag{2}$$

In regression trees the best split is calculated by using the sum of squared residuals. A decision boundary is set between two samples by calculating the mean X value between them. Afterwards the mean Y Value for the data right and left from the decision boundary is calculated. The Residual r_i is calculated by subtracting each data point from the mean, squaring the difference and adding the values. [13], [14]

If there are features X and labels or regression values Y the first step is to decide which feature should be the root split. This is done by using cross validation. In cross validation every possible combination is tried out and scores are given to each combination. The scores are given by the

calculating the Gini impurity or the sum of squared residuals. Afterwards the best combination argmin(r) is chosen. After that is decided the algorithm chooses the best splitting value that separates the data best according to the chosen feature. This is also done with cross validation. Next the split is set and the algorithm starts recursively again with the two child nodes. If a child node has an impurity or residual of 0 i.e. the data can not be separated further, the node is turned into a leaf. This continues until a new split does not improve the residual or impurity of the tree. With that approach to stopping Decision trees overfit the data because they only stop if the data can not be split further. So they are low in bias and high in variance. That means they make good predictions on the data they where trained on but do poorly on new data. The two other algorithms explored in the next sections try to resolve this issue.

4 Principles of Random Forest

The Random Forest algorithm tries to prevent overfitting by averaging over many results. To achieve that Bootstrapping and Bagging are used. Bootstrapping is a technique from statistics in which multiple samples of data are chosen. Afterwards the calculation or algorithm is applied and the results are averaged over all data samples. With enough random samples the result converges to the real value with less computation. In Bagging this idea is applied to statistical learning models. Multiple models are trained on different datasets and the average result is calculated. By using Bagging it is possible to reduce the variance in a statistical model. This happens because each model only trains on parts of the data so each model can not overfit on all of the data. By averaging over all results afterwards the predictions in the test dataset are better. [11], [13], [15]

The principle of random forest is that multiple trees on different samples of the data are created. To predict the labels Y for new data the prediction results of all trees are averaged. For regression tasks it is possible to use the average of all predicted results, for classification tasks usually the majority vote is taken. [13], [15]

It is possible that a feature X is strongly correlated to the label Y. The effect would be that a majority of the sub trees are equal because this feature X is always chosen in cross validation with the same split. That would result in uniform trees and therefor would result in a greater variance. To avoid this the features are randomly picked for each tree. Of all possible features f a subset of features $\hat{f}$ is picked, of which to construct the tree. The number of chosen features for the sub trees $\hat{f}$ is a hyperparameter which should be initialized with roughly $\sqrt{\#f}$. In practise multiple values would be tested for $\hat{f}$ and the best would be taken. [13], [15]

5 Principles of XGBoost

The XGBoost algorithm was developed in 2016 by Chen and Guestrin and is a binary tree algorithm. It also tries to prevent overfitting by using similar but yet different methods than Random Forest. In contrast to Random Forest, XGBoost prevents overfitting by adding results of many different trees and does not average over them. The individual trees are summed and scaled evenly by a learning rate. The prediction $\hat{y}$ is calculated by the formula below. T is the

set of trees, $T(x)$ is an individual tree and lr is an hyperparameter for the learning rate. [16]

$$\hat{y} = \sum_{k=1}^{\#T} (lr \cdot T_k(x)) \tag{3}$$

The trees itself are formed by using cross validation. To cross validate the possible best splits on a dataset the first step is to decide which feature to use. This can be done by multiple methods. One is cross validating over different possible trees with different features like in the CART algorithm. Another is calculating the correlation coefficient and choosing the feature with the strongest correlation to y. Afterwards the split is chosen by cross validation. The metric used to compare splits and to decide whether to stop splitting is the gain. For that the residual and similarity of each leave is necessary. The residual is calculated by $y - \hat{y}$. The similarity score $s(a)$ is calculated by summing up the squared residuals r of every datapoint d and dividing them by the number of datapoints $\#d$ and a regularization parameter λ. This parameter can be changed to further avoid overfitting of the data.

$$\text{Similarity=} \frac{(\sum r)^2}{\#d + \lambda} \tag{4}$$

Now the gain can be calculated by $Left_{similarity} + Right_{similarity} - Root_{similarity}$.
The argmax(gains) is chosen. [16]

These steps can be repeated for multiple features and splits until a maximum tree height is reached or till the gain calculated is negative for all possibilities. This maximum tree height also is an hyperparameter. The next step is pruning the trees. Pruning is a way to prevent overfitting by cutting back on splits. The decision is made by calculating gain-γ for every split. If the result is negative, the split will be removed. If a split higher in the tree is negative but a lower split is positive the tree will stay as it is. The decision when to split can be influenced by the hyperparameter λ. If λ is larger the threshold when to delete a split is lower. The optimal value for λ can be calculated by cross validating. [16]

6 Benefits and Problems

In general CART algorithms have the advantage of producing decision trees which can be graphically displayed. With that the decisions made by the algorithm can be shown and explained even to non technically persons which can be important. If a decision was wrong or someone needs to know why this decision was made it is possible to show. Also because they assign values to each decision the probability can be calculated which further supports the understanding of the decisions made. [2]

But CART also has disadvantages especially when it comes to classifying new data. Because there can be infinite splits the algorithm overfits the training data very fast. When the algorithm overfitted on the training dataset it tends to perform bad on the test dataset. This issue is called the bias variance trade-off. [2]

The advantages of being good for visual representation do not apply for the other two algorithms. This is because they create sub-trees which are calculated together. That makes a human understanding of the decisions made very complicated to impossible. Yet further comparison will

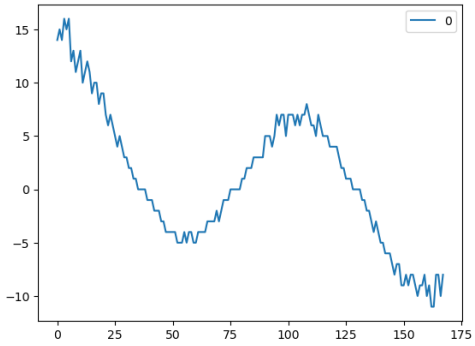
only focus on Random Forest and XGBoost because the CART algorithm overfits training data which results in inferior results. The advantages of Random Forest is that it prevents overfitting by averaging over the results. Also it reaches high accuracies and performs well even if huge parts of the data are missing. Downsides are that the algorithm is more complex and more time consuming than basic decision trees. Also the algorithm requires more computation than CART so more resources are needed. [2], [6]

Advantages of using XGBoost are that the algorithm tends to prevent overfitting the training data if the data is low in noise, so it performs good on the test set. Also XGBoost can handle missing values. On the other side it is a complex algorithm which requires a good understanding and longer training time. Also if the data is noisy it is possible that overfitting occurs. [2]

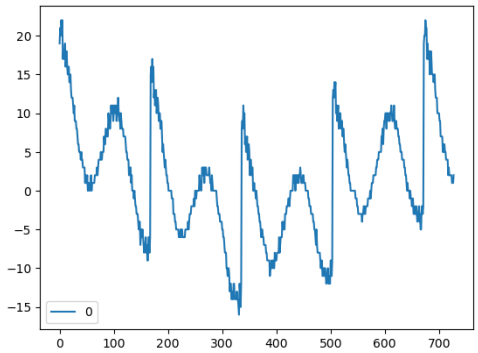
7 Synthetic Data Generation

The data which is used to compare these algorithms is synthetically generated. It resembles the deviation of calls over a year. The data is generated synthetically for multiple reasons. One is to ensure privacy of customers. Other reasons are that the data is easier to manipulate which is an advantage when modelling scenarios which there is not much data for. Another advantage is that the data is free of noise which is present in real world data or the noise can be added if needed.

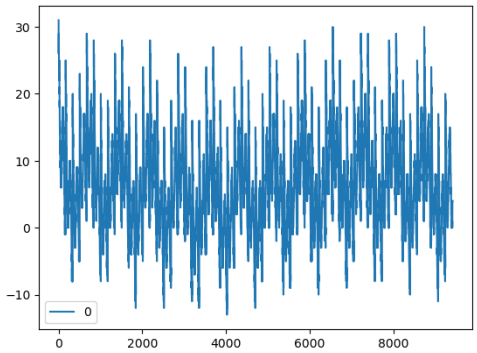
The data is generated by adding multiple functions to create the distribution. The principle of the equation can be described by $F(t) = W(t) \cdot a + M(t) \cdot b + Y(t) \cdot c$. The variables a,b,c are random numbers of different ranges. W(t) adds trends of weekdays. M(t) adds monthly trends and Y(t) are the yearly trends. The random numbers a-c are meant to scale the different results from the functions to introduce randomness and to vary the functions over the year to mimic real world behaviour better. The functions W(t),M(t),Y(t),F(t) are sin or cos functions because there are certain trends in real world call distributions that repeat over time. In these graphs they take the prior function as input to show how the data is formed.



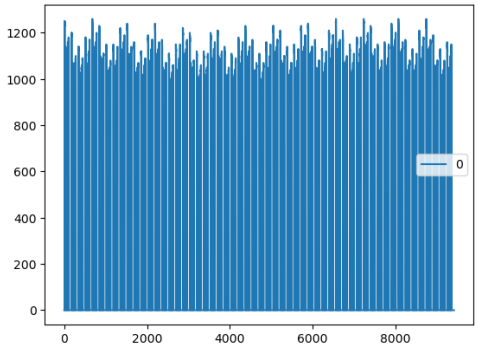
(a) $W(t) = (8 \cdot \sin(t + 25) \cdot 0.06) + 10 - t^{0.5}x) \cdot a$



(b) $M(t) = W(t) + (6 \cdot cos(2\pi \cdot (t/728)) + 0.4) \cdot b$



(c) $Y(t) = M(t) + (2 \cdot \cos(0.2\pi \cdot (t/728)) + 5) \cdot c$



(d) Non-working hours = 0, $F(t) = Y(t) \cdot 10 + 1000$

Figure 1: Functions combined to create data

The W(t) function degrades over the first half of the week and has a peek again at the end. This is because after the weekend people call to get help with the problems that occurred at the weekend and before the new weekend they have more time and want to resolve present issues. The monthly trend M(t) is a rise at the beginning of each month. This is chosen for two reasons. One is that usually salary is paid at the beginning of each month so that new devices are bought at this time and issues occur. The other reason is that if the customer has not paid three months in a row the internet connection is turned of. This happens mostly at the end of the month so by the beginning of the next month this is noticed and the call center gets an increase in calls. The seasonal trends Y(t) peaks in the winter. This is because more time is spend inside and with that technology is more often used and errors occur more often or are more often noticed. The F(t) function cuts out the non working hours i.e. weekends and before 8am and after 4pm. Also it multiplies the values by 10 and adds 1000. This has two reasons: The first reason is that calls around 1000 mimic the real world data better. The other reason is that in practise the tree algorithms work better with larger numbers. The $\cdot 10$ is meant to scale the trends to the new size.

For the comparison afterwards multiple stages of noise are added to the data.
The random.uniform(1 - n , 1 + n) function returns random values between the two numbers given. For this the F(t) graph is multiplied by this function. The n is increased with each try and the effect of increasing noise on the data is measured.

8 Methodology

The practical comparison of the algorithms is done in an experiment. In this Random Forest and XGBoost shall be compared for prediction performance on a synthetic Time Series dataset. As a metric the root-mean-squared-error will be used on the test dataset.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(Y - \hat{Y})^2} \tag{5}$$

The advantage of the RMSE is that because of the squaring negative and positive errors does not cancel each other out and the values returned are the directly interpretable as the average amount of wrong predicted calls. The datasets are split into train and test splits. For the modelling of the graphs and decisions on the machine learning algorithms the last four weeks of the first year where used with the rest of the year as training data. For the final comparison this will be changed to avoid a bias by making decisions that overfit the data for this month. The final algorithm will be trained on one whole year and predict the first month on the next year, data that was not influenced or optimized during preparation.

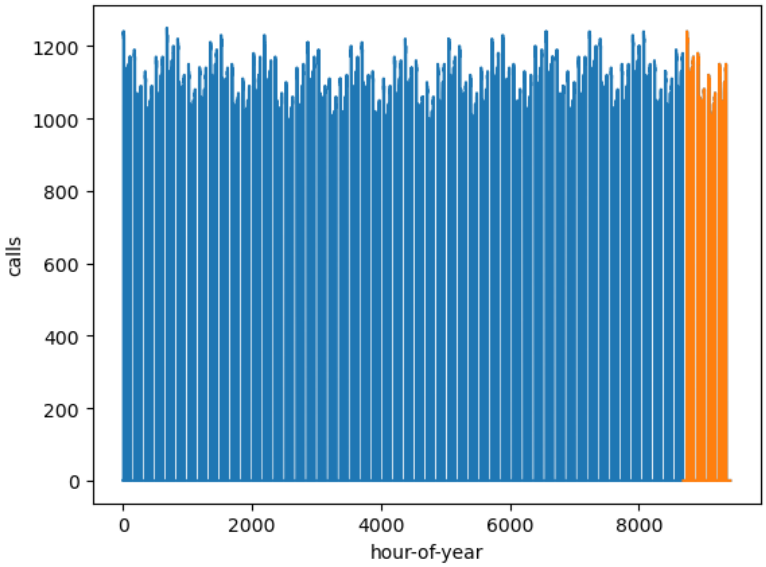
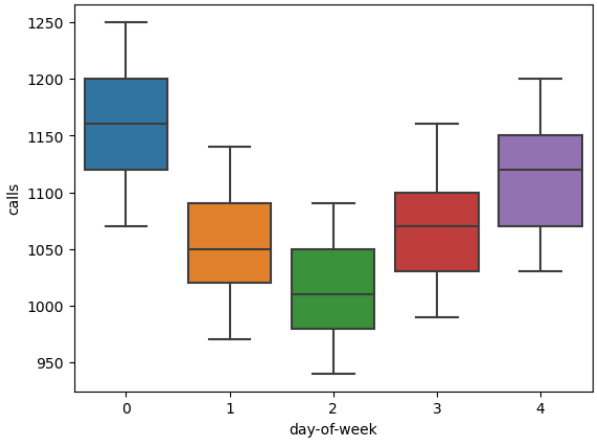
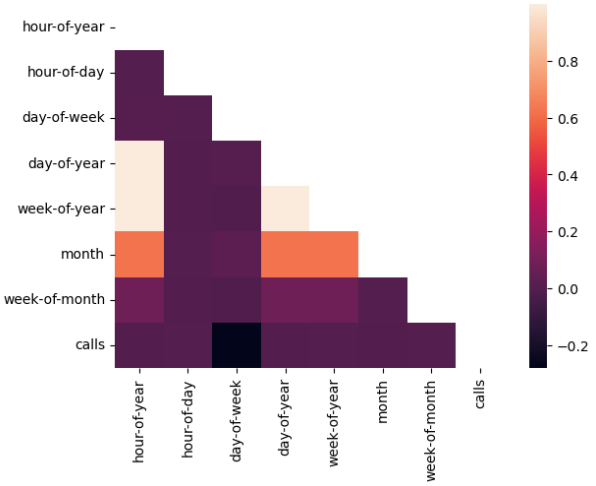


Figure 2: Train test split

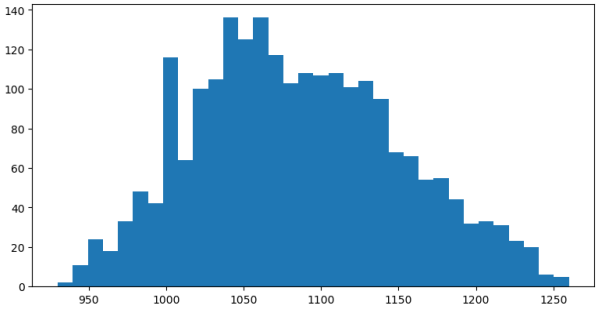
Out of the time data there are now features created to be used by the algorithms. The final feature are calls, hour-of-year ($hy = t$), hour-of-day ($hd = t \mod 24$), day-of-week ($dw = (t \mod 168)//24$), day-of-year ($dy = t//24$), week-of-year ($wy = t//168$), month ($m = (x \mod 8736)//728$) and week-of-month ($wm = (x \mod 8736) \mod 728)//168$). // resembles division with rest like in the python programming language. For a better understanding of the data there is the distribution of calls and a boxplot of the weekly trends. The days start with a 0 (Monday) and go till 4 (Friday). The heat map shows the correlation of the features that were created.



(a) Calls per day of week



(b) Pearson correlation matrix as heatmap



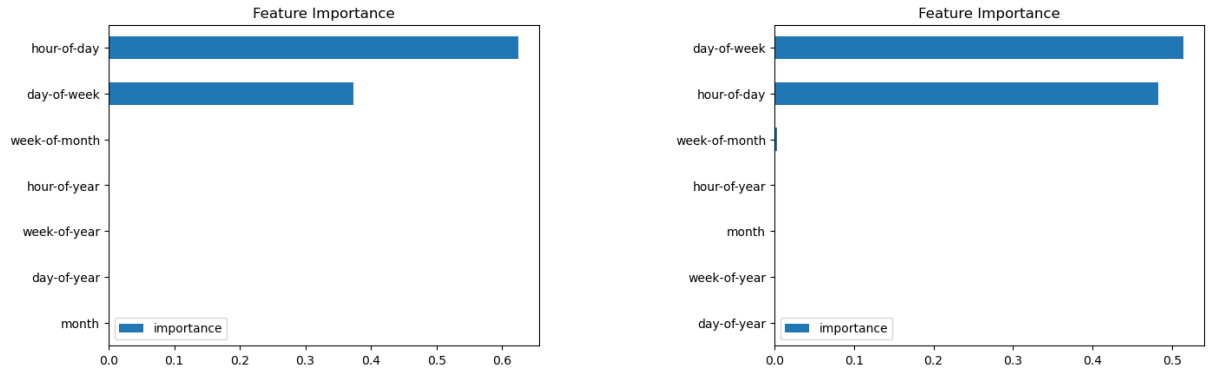
(c) Histogram for call distribution

Figure 3: Data exploratory analysis of "Seasonal data"

The trend introduced by W(t) is still present in the final data as one can see in the plot (a). The standard deviation is quite similar on all days but the mean is very different. The call distribution histogram (c) shows that like in reality the call could be roughly estimated by a Gaussian bell curve. The feature heat map (b) shows that most of the features are only very weakly correlated with the calls. The only exception is the day-of-week feature. Other features show a correlation very high correlation with each other, but not with the calls feature. This means they are dependant and could be removed at least if the correlation is 1. But because CART based algorithms does not perform worse with linear dependant features they will be left. Another reason not to remove features is that more complex algorithms like XGBoost or Random Forest might be able to pick up correlations that a mathematical calculation like the Pearson correlation matrix is not able to pick up. The implementation of XGBoost and Random Forest used for the prediction was the standard implementation of sklearn without any hyperparameter tuning except for setting a random seed to 42. The algorithms where tested on noise ranging from $n \in \{0, 0.05, ..., 0.4\}$.

9 Results and discussion

After training the algorithms the following feature importance could be evaluated.



(a) Random Forest feature importance

(b) XGBoost feature importance

Figure 4: Feature importance due to algorithms

For both algorithms day-of-week and hour-of-day where the most important features to predict upon. But XGBoost used the day-of-week more and Random Forest the hour-of-day. Also XGBoost is the only algorithm that uses the week-of-month feature slightly.

Without noise the results in both algorithms where quite promising. In figure 5 there is the best overall prediction displayed. The blue lines are the true values and the dotted orange line is the prediction. The RMSE of Random Forest for 0 noise was 10.46 so on average the prediction was off by around 10 calls which is a pretty good result when having a standard deviation of 68 calls, a mean of 1085 calls and only 1 year of data.

Both algorithms where able to pick up the difference between working and non-working hours. With increasing noise there where some outliers where the algorithms predicted very few calls in non-working hours. For lower noises both algorithms worked similarly good with Random Forest being slightly lower in RMSE the whole time.

With increasing noise XGBoost seems to have problems to identify trends in the data and predict on new data. This can be seen in figure 6. The error of XGBoost jumps up and down where as the Random Forest error nearly linear rises except for one outlier near 0.

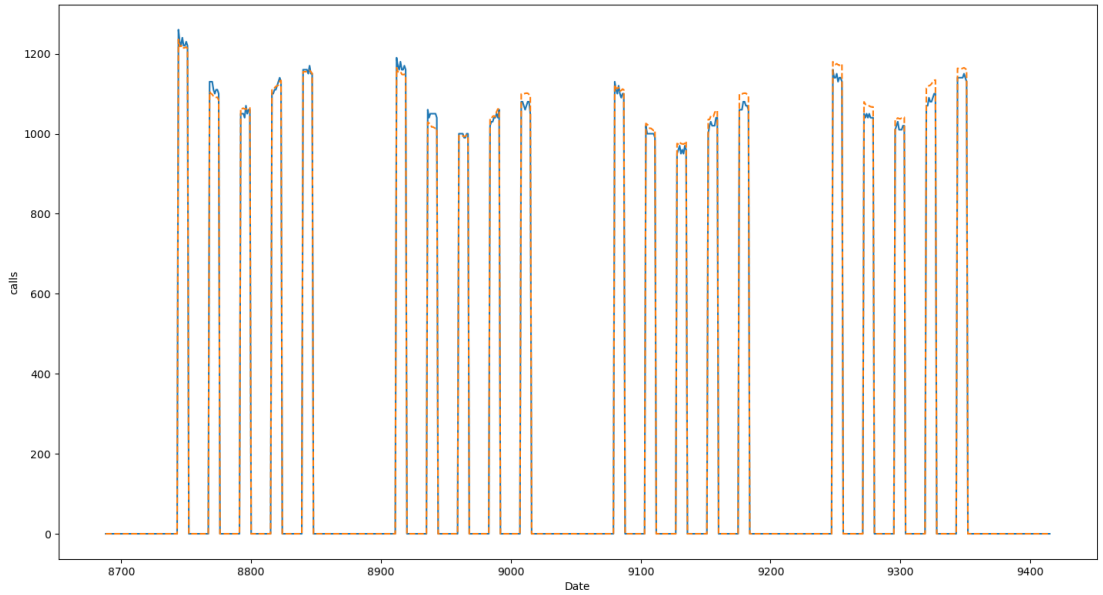
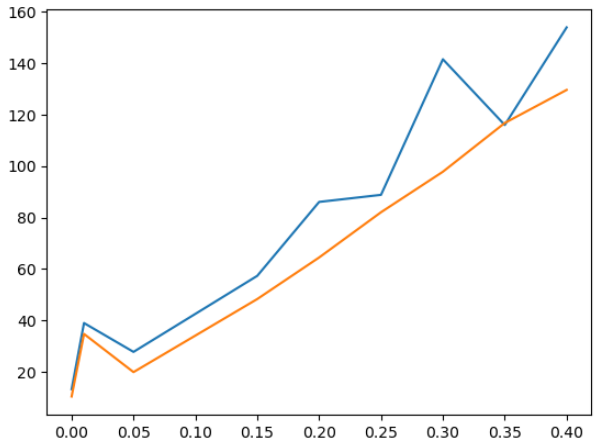


Figure 5: Random Forest prediction

Noise	XGBoost	Random Forest
0	13.29	10.46
0.05	27.80	19.94
0.1	39.02	34.78
0.15	57.37	48.35
0.2	86.11	64.51
0.25	88.86	82.1
0.3	141.58	97.85
0.35	116.02	116.8
0.4	153.97	129.66

(a) Noise RMSE table



(b) Noise RMSE graphed

Figure 6: Noise RMSE

Like presumed results suggest that there is a negative correlation between noise and the prediction accuracy. The more important result is that Random Forest seems to be more stable with noisy data. At all points of the experiment the RMSE of Random Forest was lower. The difference in RMSE increased with increase in noise. An exception was a noise of 0.35 which is an outlier and therefor it can be ignored. The reason for that could be that through Bootstrapping and Bagging the Random Forest algorithm averages over the data. The noise seems to be better ignored as a result.

This comparison focused only on comparing the algorithms on one dataset with looking only on one variable the noise. Therefor the results can not be universally applied. During developing the curves while changing variables multiple things could be noticed. Depending on for example the amplitude of the curve the RMSE of the algorithms varied strongly with XGBoost sometimes giving better results. So the effects of parameters in a curve on the algorithms trained could be a

topic for future work. Also in general comparing more different Time Series would be needed to verify the claims made by this paper on the correlation between noise and prediction. These were not examined here due to the length limits on this paper. Another potential source for errors and further investigation would be what attributes synthetical Time Series-data needs to represent real world Time Series data. Due to the periodic nature of the trigonometric functions used there might be potential differences to real world data which were not captured in this experiment.

Another topic for future research would be how the algorithms RMSE changes with optimized hyperparameters. Potentially one of the algorithms works better with optimized hyperparameters. So all the claims made by this paper would needed to be verified by more experiments with different kinds of data and with optimizing the algorithms used. To decide which algorithm is when appropriate to use future research must compare the effects of more variables on the prediction.

10 Conclusion

The results discussed in this paper can help to model synthetical Time Series data for future research and can help making decisions on when to implement which algorithm. The generation of own Time Series-data and the testing of the variables discussed in the Results and Discussion section can help to make the decision which algorithm to pick.

Just from the results of this experiment the "out of the box implementation" of Random Forest seems to work slightly better on Time Series-data and is more resistant to noise in the data. This could be due to the bagging and bootstrapping aproach of Random Forest.

Literatur- und Quellenverzeichnis

[1] R. Adhikari and R. K. Agrawal, "An introductory study on time series modeling and forecasting," *CoRR*, vol. abs/1302.6613, 2013. arXiv: 1302.6613. [Online]. Available: http://arxiv.org/abs/1302.6613.

[2] B. Akkaya and N. Çolakoğlu, "Comparison of Multi-class Classification Algorithms on Early Diagnosis of Heart Diseases," Sep. 26, 2019.

[3] Santhanam, "Application of CART Algorithm in Blood Donors Classification," *Journal of Computer Science*, vol. 6, no. 5, pp. 548–552, May 1, 2010, ISSN: 1549-3636. DOI: 10.3844/jcssp.2010.548.552. [Online]. Available: http://www.thescipub.com/abstract/10.3844/jcssp.2010.548.552 (visited on 07/31/2023).

[4] S. Markovic, J. L. Bryan, R. Rezaee, *et al.*, "Application of XGBoost model for in-situ water saturation determination in Canadian oil-sands by LF-NMR and density data," *Scientific Reports*, vol. 12, p. 13 984, 1, 1 Aug. 17, 2022, ISSN: 2045-2322. DOI: 10.1038/s41598-022-17886-6. [Online]. Available: https://www.nature.com/articles/s41598-022-17886-6 (visited on 07/31/2023).

[5] Y. Y. Liu, M. Yang, M. Ramsay, X. S. Li, and J. W. Coid, "A Comparison of Logistic Regression, Classification and Regression Tree, and Neural Networks Models in Predicting Violent Re-Offending," *Journal of Quantitative Criminology*, vol. 27, no. 4, pp. 547–573, Dec. 1, 2011, ISSN: 1573-7799. DOI: 10.1007/s10940-011-9137-7. [Online]. Available: https://doi.org/10.1007/s10940-011-9137-7 (visited on 07/31/2023).

[6] S. Krstanovic and H. Paulheim, "Ensembles of Recurrent Neural Networks for Robust Time Series Forecasting," in *Artificial Intelligence XXXIV*, M. Bramer and M. Petridis, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2017, pp. 34–46, ISBN: 978-3-319-71078-5. DOI: 10.1007/978-3-319-71078-5_3.

[7] R. Kumar, P. Kumar, and Y. Kumar, "Analysis of Financial Time Series Forecasting using Deep Learning Model," in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Jan. 2021, pp. 877–881. DOI: 10.1109/Confluence51648.2021.9377158.

[8] S. L. Salzberg, "C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993," *Machine Learning*, vol. 16, no. 3, pp. 235–240, Sep. 1, 1994, ISSN: 1573-0565. DOI: 10.1007/BF00993309. [Online]. Available: https://doi.org/10.1007/BF00993309 (visited on 08/14/2023).

[9] J. R. Quinlan, "15 - LEARNING EFFICIENT CLASSIFICATION PROCEDURES AND THEIR APPLICATION TO CHESS END GAMES," in *Machine Learning*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Eds., San Francisco (CA): Morgan Kaufmann, Jan. 1, 1983, pp. 463–482, ISBN: 978-0-08-051054-5. DOI: 10.1016/B978-0-08-051054-5.50019-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080510545500194 (visited on 08/14/2023).

[10] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Taylor & Francis, Jan. 1, 1984, 372 pp., ISBN: 978-0-412-04841-8.

[11] S. L. Crawford, "Extensions to the CART algorithm," *International Journal of Man-Machine Studies*, vol. 31, no. 2, pp. 197–217, Aug. 1, 1989, ISSN: 0020-7373. DOI: 10.1016/0020-7373(89)90027-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020737389900278 (visited on 07/31/2023).

[12] E. Laber and L. Murtinho, "Minimization of Gini Impurity: NP-completeness and Approximation Algorithm via Connections with the k-means Problem," *Electronic Notes in Theoretical Computer Science*, The Proceedings of Lagos 2019, the Tenth Latin and American Algorithms, Graphs and Optimization Symposium (LAGOS 2019), vol. 346, pp. 567–576, Aug. 30, 2019, ISSN: 1571-0661. DOI: 10.1016/j.entcs.2019.08.050. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S157106611930101X (visited on 08/14/2023).

[13] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning: With Applications in Python* (Springer Texts in Statistics). Cham: Springer International Publishing, 2023, ISBN: 978-3-031-38746-3 978-3-031-38747-0. DOI: 10.1007/978-3-031-38747-0. [Online]. Available: https://link.springer.com/10.1007/978-3-031-38747-0 (visited on 08/11/2023).

[14] C. Yu and W. Yao, "Robust linear regression: A review and comparison," *Commun. Stat. Simul. Comput.*, vol. 46, no. 8, pp. 6261–6282, 2017. DOI: 10.1080/03610918.2016.1202271. [Online]. Available: https://doi.org/10.1080/03610918.2016.1202271.

[15] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 1, 2001, ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324 (visited on 08/17/2023).

[16] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 13, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785. arXiv: 1603.02754 [cs]. [Online]. Available: http://arxiv.org/abs/1603.02754 (visited on 07/31/2023).

Appendix

data.ipynb

```
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import random as rnd
rnd.seed(42)

visualize = False

# wochentag trend
xweekly = []
yweekly = []
for i in range(0, 168):
    xweekly.append(i)
    yweekly.append(0)
xweekly = np.array(xweekly)
yweekly = np.array(yweekly)

wsin = (8* np.sin(np.array(xweekly+25) * 0.06) + 10 -
xweekly ** 0.5)

for i in range (0,7):
    i = i * 24
    for j in range(0,24):
        yweekly[i + j] =  wsin[i+j] * rnd.uniform(0.9, 1.1)

if visualize:
pd.DataFrame(yweekly).plot()

# Monthly trends, create arrays
xmonthly = []
ymonthly = []
duration = 8736 * 1 + 728 - 48
for i in range(0, duration): #8736 = 1 Jahr
    xmonthly.append(i)
    ymonthly.append(0)
xmonthly = np.array(xmonthly)
ymonthly = np.array(ymonthly)

# Monthly trends function
msin = 6* np.cos(2*np.pi * (xmonthly / 728)) + 0.4 # Edit *15


# Add Trend
```

```
for i in range (0,duration // 168):
    i = i * 168
    for j in range(0,168):
        ymonthly[i + j] = yweekly[j] + msin[i + j] \
            *rnd.uniform(0.9, 1.1)


if visualize:
    pd.DataFrame(ymonthly[:728]).plot()

# Yearly trends, create arrays
xseasonal = xmonthly.copy()
yseasonal= ymonthly.copy()
# Seasonal trends function
ssin = 2 *np.cos(0.2*np.pi * (xseasonal / 728)) + 5


for i in range(0, len(yseasonal)):
    yseasonal[i] = ymonthly[i] + ssin[i] * rnd.uniform(0.9,1.1)

if visualize:
    df = pd.DataFrame(yseasonal)
    df.plot()

# Cut Weekend and Non working hours
yseasonal = yseasonal.copy() * 10 + 1000
xseasonal = xseasonal.copy()
for i in range(0, len(yseasonal)):
    #cut non working hours
    if xseasonal[i] % 24 < 8 or xseasonal[i] % 24 >= 16:
        yseasonal[i] = 0

    #cut weekend
    if (xseasonal[i] % 168) > 120:
        yseasonal[i] = 0

    if yseasonal[i] < 0:
        yseasonal[i] = 0

if visualize:
    pd.DataFrame(yseasonal).plot()

# noise
xsnoise = xseasonal
ysnoise = yseasonal
n = 0.05
for i in range(0, len(xsnoise)):
    ysnoise[i] *= rnd.uniform(1 - n, 1 + n)
```

```
# Feature creation
def create_features(x,y):
    df = pd.DataFrame()
    df['hour-of-year'] = x
    df['hour-of-day'] = x % 24
    df['day-of-week'] = (x % 168) // 24
    df['day-of-year'] = x // 24
    df['week-of-year'] = (x // 168)
    df['month'] = (x % 8736) // 728
    df['week-of-month'] = ((x % 8736) % 728) // 168
    df['calls'] = y['calls']
    return df

df = create_features(xseasonal,
pd.DataFrame(yseasonal, columns=['calls']))

if visualize:
    feature_corr = df.corr()
    mask = np.triu(np.ones_like(feature_corr))
    sns.heatmap(data=feature_corr, mask=mask)

#.hist()
if visualize:
    fig = plt.figure(figsize=(8,4))
    ax = fig.add_axes([.1,.1,.9,.9])
    ax.hist(df['calls'][df['calls'] > 0], bins=34,)

if visualize:
    sns.kdeplot(df['calls'][df['calls'] > 0])
    # Kernel density estimation of distribution

# Train test split
s_concat = create_features(x=xseasonal,
y=pd.DataFrame(ysnoise, columns=['calls'])).copy()
s_train = s_concat[:duration - 728]
s_test = s_concat[duration - 728:]

if visualize:
    sns.boxplot(x='day-of-week', y='calls',
    data=s_concat[s_concat['calls'] != 0])

if visualize:
    sns.boxplot(x='week-of-month', y='calls',
    data=s_concat[s_concat['calls'] != 0])

if visualize:
    sns.lineplot(x='hour-of-year', y='calls' , data=s_concat)
```

```
    sns.lineplot(x='hour-of-year', y='calls' , data=s_test)
s_concat[s_concat['calls'] != 0].describe()
```

models.ipynb

```
import pandas as pd
import numpy as np
import sklearn
import datetime
import matplotlib.pyplot as plt
import seaborn as sns
import random as rnd
rnd.seed(42 )

import xgboost as xgb
from sklearn.metrics import mean_squared_error
from sklearn import ensemble
%run data.ipynb
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

FEATURES = ['hour-of-year', 'hour-of-day', 'day-of-week',
    'day-of-year', 'week-of-year', 'month', 'week-of-month']
TARGET = 'calls'
X_train = s_train[FEATURES]
y_train = s_train[TARGET]
X_test = s_test[FEATURES]
y_test = s_test[TARGET]

# XGBoost
xgb_reg = xgb.XGBRegressor(seed=42)


xgb_reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],

        verbose=100
        )
y_test

# Feature importance
f1 = pd.DataFrame(data=xgb_reg.feature_importances_
, index=xgb_reg.feature_names_in_, columns=['importance'])

f1.sort_values('importance').plot(kind='barh',
title='Feature Importance')
plt.show()

pred = pd.DataFrame({'calls':[0]*len(X_train)})
pred = pd.concat([pred, pd.DataFrame(
    {'calls' : xgb_reg.predict(X_test)})],
```

```
    axis=0, ).reset_index(drop=True)
fig = plt.figure(figsize=(15,8))
ax = fig.add_axes([.1,.1,.9,.9])
ax.set_xlabel("Date")
ax.set_ylabel('calls')

['calls'].to_numpy())
ax.plot(y_test)
index = len(X_train)
ax.plot(pred[index:], '--')
np.sqrt(metrics.mean_squared_error(y_test, y_pred=pred[index:]))

#Random Forest
rf_reg = sklearn.ensemble.RandomForestRegressor(random_state=42)
FEATURES = ['hour-of-year', 'hour-of-day', 'day-of-week',
    'day-of-year', 'week-of-year', 'month', 'week-of-month']
TARGET = 'calls'

rf_reg.fit(X_train, y_train,)
y_test

# Feature importance
f1 = pd.DataFrame(data=rf_reg.feature_importances_
, index=rf_reg.feature_names_in_, columns=['importance'])

f1.sort_values('importance').plot(kind='barh',
title='Feature Importance')
plt.show()

pred = pd.DataFrame({'calls':[0]*len(X_train)})
pred = pd.concat([pred, pd.DataFrame(
    {'calls' : f_reg.predict(X_test)})],
    axis=0, ).reset_index(drop=True)
fig = plt.figure(figsize=(15,8))
ax = fig.add_axes([.1,.1,.9,.9])
ax.set_xlabel("Date")
ax.set_ylabel('calls')

ax.plot(y_test)
ax.plot(pred[len(X_train):], '--')
np.sqrt(metrics.mean_squared_error(y_test,
y_pred=pred[len(X_train):]))

# Plot rmse
y_xgb_rmse = [13.29, 27.80, 39.02, 57.37,
    86.11, 88.86, 141.58, 116.02, 153.97]
y_rf_rmse = [10.46, 19.94, 34.78, 48.35,
    64.51, 82.1, 97.85, 116.8, 129.66]
```

```
x_rmse = [0, .05, .01,.15,.2,.25,.3,.35,.4]
sns.lineplot(x=x_rmse, y=y_xgb_rmse, markers=True)
sns.lineplot(x=x_rmse, y=y_rf_rmse, markers=True)
np.median(np.absolute(np.array(y_xgb_rmse) - np.array(y_rf_rmse)))
```