Boosting public water
Natural Computing 2018

Bauke Brenninkmeijer - s4366298
Ties Robroek - s4388615

June 20, 2018

Contents

List of Tables

1 Introduction

In this work we sought to evaluate the fitness of natural computing related classification algorithms such as boosting and bagging as alternatives to the state of art, which are currently almost always neural networks, for anomaly detection. More specifically, we implemented boosting and bagging to detect anomalies in the quality of water. These anomalies are measured by Thüringer Fernwasserversorgung, a water treatment facility from Germany. This work has been produced for the course Natural Computing, Radboud University. This course has shown us many different algorithms based on nature and genetics. Two lectures described different Ensemble methods. While perhaps not strictly natural computing, these methods combine multiple weaker classifiers into one big stronger classifier. We wanted to discover the potential of these methods as they were new to us and we have not used them before. In the end we have discovered the systems Adaboost, XGBoost, bagging and ensembling between these.

2 Specific goals of the work

The Genetic and Evolutionary Computation Conference (GECCO) is an international annual genetic programming conference. Researchers from all over the world attend to show their work in the field of genetic and evolutionary computation. Some of the features of this conference are the so-called Gecco Challenges. These challenges are organized as classic Machine Learning Competitions. Teams have to come up with an algorithm that performs best on a specific task. These tasks tend to be extremely challenging, which is why exact mathematical solutions are usually impossible. The Gecco challenges ask for well-designed genetic algorithms that can sufficiently fulfill the task.

Figure 1: The Genetic and Evolutionary Computation Conference (GECCO) is the pinnacle of genetic and evolutionary computation.

2.1 The challenge

SPOTSeven Lab is a research group that has hosted numerous challenges for Gecco. Their challenges are fashioned with industrial and real-world applications in mind. Good solutions may be considered by their industrial partners for future implementation. SPOTSeven is hosting an anomaly detection task this year. We have chosen to tackle this challenge. The SPOTSeven Industrial Challenge 2018 regards water quality. The goal is to develop an event detector to accurately predict any kinds of changes in a time series of drinking water composition data. Water is an immensely important resource. It is a requirement for all known depictions of life. Any changes in the quality of drinking water might thus have a profound impact on us and our environment. The challenge asks for an anomaly detection system for our drinking water. The data is collected by the challenge's main industrial partner Thüringer Fernwasserversorgung (TFW). This is a German public water company that operates more than 60 dams and reservoirs. The data is sources from their over 550 km of bulk water transport network. Data is collected using a system of sensors that collect readings concerning the most important water quality indicators. The data is collected at different stations near the outflow of a waterworks.

In this day and age neural networks are immensely popular. The deep learning boom has spiraled their popularity to unprecedented heights. They do not dominate every field in terms of performance though. In Machine Learning competitions such as those hosted on Kaggle boosting methods are immensely succesful. Especially XGBoost has reached levels of fame for frequently facilitating winning entries in such competitions. This is why we have opted to explore such methods in this work. We will be looking at the ADABoost and XGBoost systems as well as some bagging methods. We will provide in-depth performance analysis on all of our implementations. The competition uses F1-

Column name	Description
Time	Time of measurement, given in following format: yyyy-mm-dd HH:MM:SS
Tp	The temperature of the water, given in C
Cl	Amount of chlorine dioxide in the water, given in mg/L (MS1)
pH	PH value of the water
Redox	Redox potential, given in mV
Leit	Electric conductivity of the water, given in mS/cm
Trueb	Turbidity of the water, given in NTU
Cl_2	Amount of chlorine dioxide in the water, given in mg/L (MS2)
Fm	Flow rate at water line 1, given in m3/h
Fm_2	Flow rate at water line 2, given in m3/h
EVENT	Marker if this entry should be considered as a remarkable change resp. event, given in boolean

Table 1: Description of the TFW dataset

score on the test set as the primary performance metric. We will be comparing these scores and show other evaluation metrics to explain which method may be most promising for such work.

3 Related work

In this section we will be primarily exploring the methods we have implemented for this challenge. We have written sections on AdaBoost, bagging and XGBoost.

3.1 AdaBoost

Adaboost[5] has been a prominent boosting algorithm for a long time. Adaboost stands for Adaptive Boosting. This adaptive measure has several serious advantages over "non-adaptive" boosting as described later in this subsection. Boosting algorithms combine weaker algorithms to create a stronger algorithm. These smaller algorithms are commonly referred to as weak learners. Weak learners do generally not score very high. This is offset by their general simplicity; they take relatively little time to process. This flexibilty allows multiple weak learners to be chained into a stronger algorithm. This stronger algorithm is commonly referred to as the boosted classifier, given by the following formula:

$$F_T(x) = \sum_{t=1}^{T} f_t(x)$$

Where:

- F_T, the boosted classifier
- T, the amount of weak learners
- f_t, a weak learner
- x, the data object to be classified

As evidently shown the boosted classifier combines the performance of the weak learners into one more powerful classifier. This is the basic principle of boosting. The algorithm can learn the dataset via its learning iteration. This is done by minimizing the sum trained error E_t:

$$E_t = \sum_i E[F_{t-1}(x_i) + f_t(x_i)$$

Where:

- E_t, the sum trained error
- $E[]$, the error function
- F_{t-1}, the previous boosted classifier which is to be improved via the inclusion of another weak learner
- x_i, iterator over the data objects
- f_t, a weak learner that is being considered, described as:

$$f_t(x_i) = a_t h(x_i)$$

  - f_t, the weak learner to be considered
  - a_t, coefficient for the learner as to minimize the error
  - h, the base weak learner

Weak learners are considered and are assigned a coefficient. This ensured the addition of the most valuable learner per iteration. The training set is also assigned multiple weights to improve learning behaviour. Every sample is assigned a weight. This weight is determined by the current error on that sample. This allows the iterative algorithm to focus on very high error (and thus very high weight) samples. This behaviour explains the "Ada" in Adaboost, standing for Adaptive. Adaboost is not perfect. A problem it shares with other boosting algorithms is its proneness to overfitting. The adaptiveness may help to reduce overfitting on sparse datasets such as ours. Nevertheless, it will still

overfit when run for longer periods of time. One important way to combat this is to stop early. This will simply make it impossible for the algorithm to highly fit around the dataset. There are simply not enough iterations to do so. Adaboost's adaptive weights should theoretically offer great advantages for anomaly detection. The weights may allow the algorithm to very heavily focus on the anomalies. Other algorithms that do not have such measures may simply fail to prioritize these cases. Our test setup includes extensive tests to assert whether the algorithm has been trained appropriately or not.

3.2 Bagging

Bagging[2] is an ensemble method in Machine Learning. Commonly called bagging, bootstrap aggregating heavily combats overfitting while attempting to improve stability and accuracy. One of the challenges of bagging will be to effectively aggregate the algorithms. Bagging splits a large training set into multiple new training sets. These sets are samples from the main set. It is very important to note that this sampling is done with replacement. Samples may thus contain multiple representations of the same object. Classifiers are trained on these samples and are finally aggregated. Bagging can have difficulties generating top-performing entries. The bootstrap sampling solves multiple issues. It can also yield impressive accuracy improvements in some instances. Many problems, however, do not benefit from bagging accuracy-wise. Our main concern is thus whether bagging will result in a noteworthy score.

3.3 XGBoost

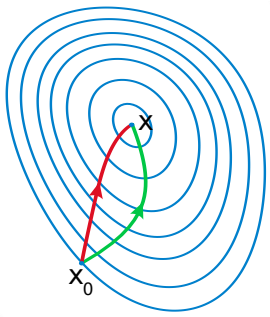


Figure 2: XGBoost uses gradient descent to slowly improve itself by choosing and optimizing classifiers.

XGBoost is an immensely popular boosting method. It is a form of gradient boosting that focuses on performance and scalability. It has been the highlight of many top-performing entries in machine learning competitions. XGBoost offers great performance on large supervised datasets. This would fit our problem perfectly.

XGBoost uses gradients to boost its trees. This is unlike AdaBoost, which does not follow a gradient. XGBoost builds on a broader subset of boost algorithms called gradient boosting algorithms. The in data science well-known practice of gradient descent is used to train the boosting classifier. Interestingly, the creators of the XGBoost algorithm do not claim that it is stronger than other boosting algorithms. XGBoost simply has the advantage of performance. This means that XGBoost approximates the "valley" of its gradient a lot faster than conventional algorithms. The name XGBoost is derived from "Extreme Gradient Boosting", with "Extreme" referencing it's time performance. XGBoost implementations are known for their scalability and perform extremely well on complex problems. In many situations XGBoost may be able to train a large model whereas other methods do not have the computing performance to generate such a model. In our case our dataset is limited. We expect XGBoost to perform quite reasonable but to not top the charts on its own.

4 Results

For our results we tried all three methods separately, as well as an ensemble of the top two approaches. We have included two baselines. In this project, we feel an false positive and a false negative are equally undesired, and thus we did not optimize for either recall or precision, but rather F1-score. This was also the main metric used by the challenge. For the results in 2, we adapted the evaluation function of the challenge to include training the network with a single pass over the data with a random split. Our results include the models RandomBooster and Falsy McFalseFace as well as all of our described models. RandomBooster is the standard template classifier provided by the challenge. It randomly classifies datapoints as positive or negative. Falsy McFalseFace is a test classifier we included early on that classifies every entry as negative. We included this to get a sense of the dataset as in anomaly detection most cases are not an anomaly. Finally we have included a test varying training set sizes for our engineered classifiers.

SUBMISSION	TP	FP	TN	FN	F1
Robroek en Brenninkmeijer - ADABoostDetector	515	4	45998	5	0.99133782
Robroek en Brenninkmeijer - BaggingDetector	462	17	45985	58	0.92492492
Robroek en Brenninkmeijer - XGBoostDetector	507	3	45999	13	0.98446602
Robroek en Brenninkmeijer - EnsembleDetector	513	5	45997	7	0.9884393
RandomBooster	246	23094	22908	274	0.02062028
Falsy McFalseFace	0	0	46002	520	0.00000000

Table 2: The results of averaged runs for all models.

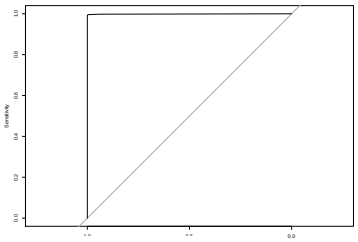
4.1 AdaBoost

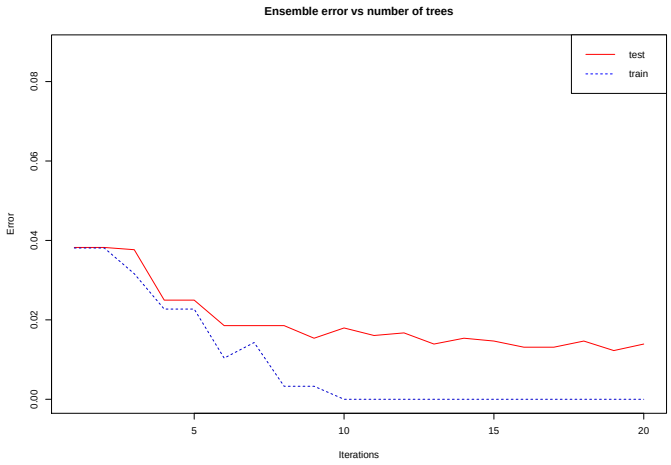
AdaBoost has a long track record of good performance, and in this case also proved to be the strongest as a sole performer. Using the Adabag[1] package in R [6], we implemented AdaBoost. We set the maximum depth of the trees, variable max-depth, to 15. We limited the amount of iterations for which boosting occurs to 20. AdaBoost is an expensive algorithm, which is why we chose for this maximum depth. The limit prevents unnecessary workload and perhaps more importantly prevents heavy degrees of overfitting.

In table 2, the specific results can be seen. With an impressive F1-score of 0.991, Adaboost performed best in our tests. In figure 3 we show the ROC curve of AdaBoost. In terms of ROC curves this is a very good result. ROC curves, however, even though they are standard metrics for machine learning classifiers, are not fit for evaluating our system. Found errors are often so small in anomaly

detection so that it is not really visible to the human eye. In figure 4 we took the square root of the error to increase visualization capacities. Plotted against the training iterations the error rate becomes clearly visible as the error rate is around 0.03. We can see a clear learning curve in the first 10 iterations, after which it remains mostly stable.

Compared to the other two approaches, Adaboost stood out in two ways. Firstly, the performance was the best and secondly, the training duration was by far the longest since, in this package, no parallelization was possible. However, predictions are fast so usage as a real time classifier is not impaired.





C curve.

Figure 4: AdaBoost is a great performer but requires a lot of time for training compared to bagging and XGBoost.

4.2 Bagging

The second method we considered was bagging. Bagging performed a bit worse in precision, but scored higher in speed. Training took mere seconds due to the parallel capacities of bagging. For bagging we used the Adabag [1] package, similar to what we did with Adaboost. The performance of bagging, however, which can be seen in table 2 was significantly lower. Bagging scored an F1-score of 0.92. Compared to the 0.99 and 0.98 of Adaboost and GXBoost, respectively, this can be regarded as a non-significant classifier for this problem, since it does not deliver the required precision and recall.

In figure 5 the square root of the bagging error is plotted against the number of trees. We can see here that the parallel working of bagging the effect of multiple trees reduces once a certain precision has been reached. The performance did see a very slight improvement, but this was so marginal we disregarded this.

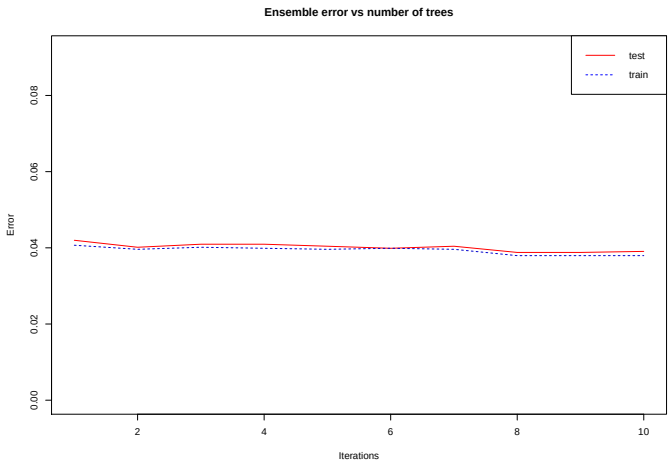


Figure 5: Bagging's error versus the number of trees squared.

4.3 XGBoost

The heavy weight XGBoost [3] did not come out on top in the challenge evaluation. With a F1-score of 0.98 it could not beat Adaboost in a single run. However, XGBoost was significantly faster than Adaboost due to its parallel capacities. For XGBoost, we used the XGBoost[4] package.

As figure 6 shows, with multiple numbers of trees, ranging from 1 to 20. Now, the accuracy becomes very good, but when calculated as F1, this still corresponds to the same F1-score as reported above. The figure does neatly show how the performance of XGBoost improves significantly with more trees.

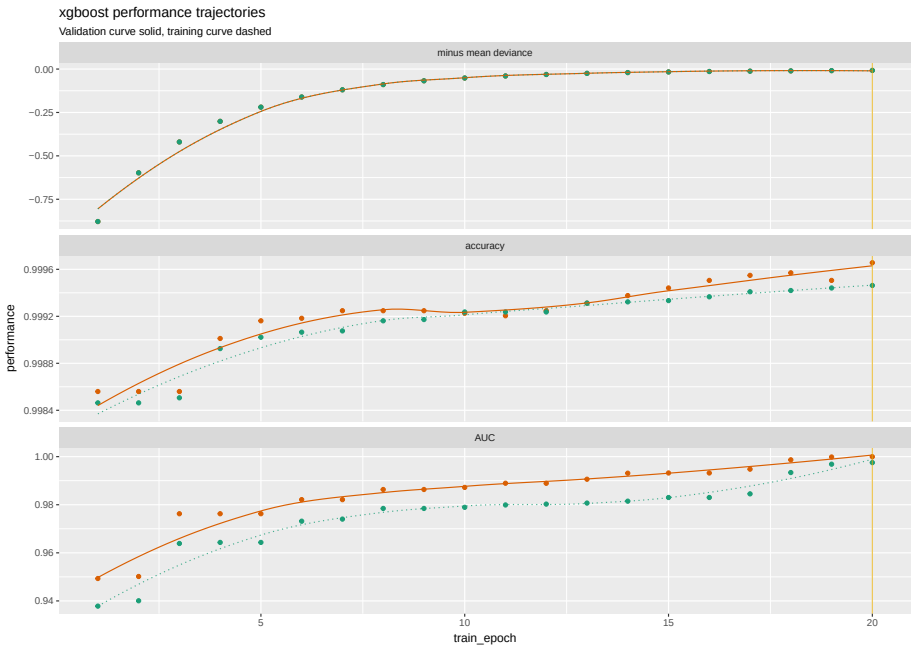


Figure 6: The minus mean deviance, accuracy and AUC of XGBoost

4.4 Ensemble

To improve on the found results, we also implemented a ensemble of the two strongest learners; Adaboost en XGBoost. We managed to improve slightly on the found Adaboost results, by taking the mean of the probabilities of the predictions and thresholding at a certain level. We cross validated 10 times, with random 2/3 for training and 1/3 for testing. With a threshold of 0.3, we achieved the best results. However, these results still didn't always improve upon Adaboost alone. This might be due to the fact that Adaboost was in general better than XGBoost, but were equally weighted in this ensemble. With an error of 0.00038 on threshold=0.3, we received a very similar ROC curve to the one of Adaboost, visible in figure 8

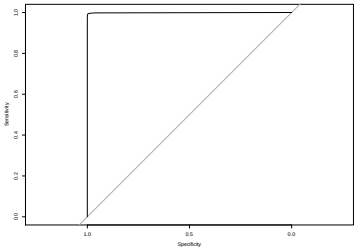


Figure 7: The ROC curve obtained via our ensemble.

The fact that bagging was insufficient turned out to be an advantage for the ensemble. Bagging does not report probabilities but only predicted classes. This is contrary to both Adaboost and XGBoost, who both also report probabilities.

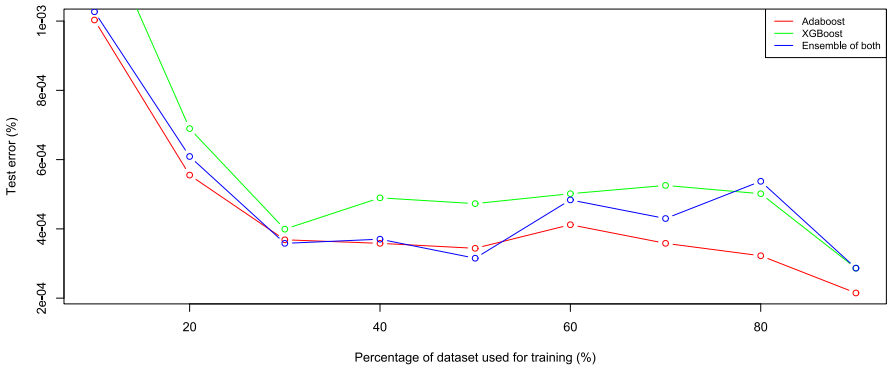


Figure 8: Performance of our classifiers with varying training set sizes.

4.5 Data size analysis

To conclude our research we finished with a data size analysis. We wrote a script that varies the amount of training data for the algorithms. Unlike the naive starting classifiers all of our engineered systems require to be trained. The more data the better tends to be the golden rule for these classifiers. This means that reducing the training data to a fraction of the total set may severely impact the quality. The effects of shrinking the training set may not be groundbreaking for the contest, but they may provide valuable insight on how the algorithm handles the data. From our personal perspective this test also tells a story about the algorithm's potential on different datasets. We can clearly see that XGBoost struggles with smaller training sets. This may be expected as the algorithm is optimized for grand scale use on big datasets. AdaBoost shows an obvious improvement with increase of the dataset size but does still perform very respectable even with just half the set left. Further research may find the collection of more data interesting as it may enhance XGBoost to beat the performance of AdaBoost.

5 Conclusion & Future work

In our project we have shown the feasability of anomaly detection using natural computing related algorithms like boosting and bagging, using Adaboost, bagging and XGBoost. Our results show these algorithms can still be very effective for modern day tasks and achieved state of the art results. Unfortunately, the timing of this assignment was too late to actually be able to submit this work to the challenge.

There are many options for future work, which we did not get to. We believe further investigation of optimal optimal parameters might increase the performance even more, closing in on an even higher F1-score. Also, the inclusion of an in depth analysis of why the bagging method did not perform as well as the others could give valuable insights. Lastly, we believe the settings of the ensemble between XGBoost and Adaboost could be improved. Just taking the mean might not be optimal, since Adaboost performs better on its own. Shifting the weight balance to Adaboost might then yield increases in accuracy. Our aim for this study was to discover the potential of these methods.

6 Contributions

The project distribution was fairly balanced, as all things should be. In the implementation created Bauke the Adaboost, XGBoost and ensemble . Ties created the bagging and adapted the evaluation function to accommodate our implementation specific needs. In the report, Ties wrote the introduction, along with the related work and specific goals of the work. Bauke wrote the results, the conclusions and the contributions. Contributions were equal in size.

7 Project sources

Our project resources can be found on github:
`https://github.com/Baukebrenninkmeijer/NaturalComputingFinal`.
All information regarding the contest can be found on the SPOTSeven website:
`http://www.spotseven.de/gecco/gecco-challenge/gecco-challenge-2018/`
This project is our final project for the Radboud University course Natural Computing (2018).

References

[1] Alfaro, E., Gámez, M., and García, N. adabag: An R package for classification with boosting and bagging. *Journal of Statistical Software 54*, 2 (2013), 1–35.

[2] Breiman, L. Bagging predictors. *Machine learning 24*, 2 (1996), 123–140.

[3] Chen, T., and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (2016), ACM, pp. 785–794.

[4] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., and Li, Y. *xgboost: Extreme Gradient Boosting*, 2018. R package version 0.71.2.

[5] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55*, 1 (1997), 119–139.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013.

[7] Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.-C., and Müller, M. proc: an open-source package for r and s+ to analyze and compare roc curves. *BMC Bioinformatics 12* (2011), 77.