FYS-STK3155 - Project 3 Autumn 2019[8]

Johannes A. Barstad, Olve Heitmann

December 18, 2019

Abstract

The project deals with implementation, application, and discussion/ consideration of the predictive performance of Decision Trees, Gradient Boosted Trees, and Random Forest on the Wine Quality Data set available through UCI's ML Repocitory. In addition we also apply the XGBoost-algorithm by Chen and Guestrin to the same data set. To estimate optimal values for hyperparameters we use techniques such as k-fold cross validation. Predicting wine quality is a difficult task as taste is the least understood of the human senses — successfully modeling quality RESULTATER

1 Introduction

Ensembles of Decision Tree models and *XGBoost* have dominated online machine learning competitions on structured, tabular data sets as long as Kaggle has been around[1].
On the mission to reduce "choice overload" in academic research, *Olson et. al* come to similar conclusions through studying 165 data sets in bio informatics:

> "Although having several readily-available ML algorithm implementations is advantageous to bio informatics researchers seeking to move beyond simple statistics, many researchers experience "choice overload" and find difficulty in selecting the right ML algorithm for their problem at hand." [14]

> "The post-hoc test underlines the impressive performance of Gradient Tree Boosting, which significantly outperforms every algorithm except Random Forest at the $p < 0.01$ level." [14]

Both academic and real life evidence clearly promotes these models as offering promising starting points for modeling tasks — both for classification and regression problems. However, both academic researchers and practitioners alike acknowledge the need to make tests on the actual data set that is subject of interest, as well as dedicating time and resources to tune *hyperparameters*[14].

In this project we aim to study these methods in detail, by implementing a subset of the methods by ourselves, applying them to real data, and then performing analysis focusing on the most characteristic aspects of each method. We have chosen to focus regression problems, and applying our implemented algorithms on the *Wine Quality Data Set*, available through UCI's Machine Learning Repository[6]. Predicting wine quality through the use of artificial intelligence and/ or data mining methods have received a lot of attention in popular science the last few years (see e.g. *Artificial Vintelligence: AI Gets Taste of Wine Industry*[18]), and we thought it would be an interesting data set to study.

In addition to applying our self-developed algorithms to the data, we also apply and analyze *XGBoost*, a specific gradient boosting implementation by Chen, T., and Guestrin, C.[2]. XGBoost is by many considered one of the leading implementations of these types of methods, and differ from regular gradient boosting machines both by system and hardware optimization and algorithmic enhancements.

In terms of structure of the paper we start of by discussing the most important aspects of the theoretical background for our covered methods, and how they relate (with the exception of the XGBoost algorithm — for details here we refer to Chen, T. and Guestrin, C.'s paper[2]). We then move on to briefly discuss where to find our implementations, as well as briefly describing the data set. In the final part we present and discuss

our results, as well as presenting a conclusion to which methods performed the best.

2 Theory

Initial comments
This theory section is a summary of what we believe are the most relevant discussions pertaining to our implementation and analysis of Decision Trees, Random Forests, and Gradient Boosted Trees. The section is based upon lecture notes provided in the course[11], chapter 8, 9, 10 and 15 in Hastie et. al[10], and various internet resources — e.g. *explained.ai*[15].
As our practical application focuses on a regression problem, we have prioritized elaborating on aspects related to solving regression problems, rather than also including classification issues and attempting to cover the topic exhaustively.

For the entirety of this section, the training data is assumed to consist of p inputs and a response variable for N observations for the entirety of this section; that is (x_i, y_i) for $i = 1, 2, \ldots, N$ with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$

2.1 Decision Trees

Decision Trees are supervised learning methods used for classification and regression problems. They work by partitioning the feature space into mutually exclusive and collectively exhaustive regions, and making predictions by assigning a certain class or regression value to each region — every observation that belongs to a given region then receives the same prediction. In this paper we focus on *CART* implementation of trees — other noteworthy methods include *ID3* and later versions, *C4.5* and, *C5.0*. For a discussion on the main differences see e.g. Hastie et. al. chapter 9[10].

Overarching architechture
Decision Trees consist of a *root node*, a set of *interior nodes*, and the final *leaf nodes (leaves)*. These entities are connected by a set of *branches*.

The leaf nodes contains the predictions.

2.1.1 Making predictions and fitting Regression Trees

For regression problems with sum of squared errors loss, predictions are made according to the following procedure:

1. Partition the predictor space (i.e. possible values of x) into M distinct and non-overlapping regions $R_1, R_2, \ldots, R_M$

2. Let c_m be the mean of the response values for the training observations in R_m for every observation that falls into the region, i.e. $c_m = ave(y_i | x_i \in R_m)$. Predictions are then made according to the following equation:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

where $I(\cdot)$ denotes the indicator function, returning 1 when the statement $\cdot$ is true, and 0 otherwise.

The quality of fit for a regression tree thus depends solely upon the construction of the regions $R_1, R_2, \ldots, R_M$. To ensure easy interpretation these regions are chosen to take the shape of *high dimensional rectangles*, or *boxes*.

Recursive binary splitting
It is generally computationally infeasible to consider every possible partition of the predictor space into M boxes. The common strategy is to take a *greedy*, top-down approach called *recursive binary splitting*. Beginning at the root (top) of the tree, recursive binary splitting works by

1. Selecting a single predictor x_j and a cutpoint s splitting the predictor space into two regions R_1, R_2, to minimize MSE, that is

$$R_1 = \{x_i | x_j < s\}$$

$$R_2 = \{x_i | x_j \geq s\}$$

$$\min_{x_j, s} \sum_{i: x_i \in R_1} (y_i - c_1)^2 + \sum_{i: x_i \in R_2} (y_i - c_2)^2$$

2. Repeating the process within each of the resulting regions, stopping when we reach some stopping criterion, e.g. when each terminal node has fewer than some minimum number of observations

Binary splitting is preferred over multiway splits as multiway splitting often end up fragmenting the data too quickly, leaving insufficient number of observations the next level down (Hastie et. al, page 311)[10].

2.1.2 Pruning Decision Trees

To reduce the probability of *overfitting*, decision trees are typically *pruned*. The fundamental idea

is to grow a large tree T_0, and then prune it back in order to obtain a sub-tree. This pruning process is done by modifying the cost function with a *complexity penalty*.

Cost complexity pruning
The cost complexity criterion is defined by

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

where $Q_m(T) = \frac{1}{N_m}\sum_{x_i \in R_m}(y_i - \hat{c_m})^2$, and $N_m = \#\{x_i \in R_m\}$ where $\#\{\cdot\}$ denotes the number of elements in the set where the $\cdot$ condition is true. $|T|$ is the number of terminal nodes of the tree T.

The hyper-parameter α controls the trade-off between the sub-tree's complexity and its fit to the training data — for $\alpha = 0$ the criterion is equivalent to the normal loss function, and the solution is the full tree T_0. However as we increase α from 0, branches get pruned from the tree by *weakest link pruning*, and we get a finite sequence of sub-trees containing T_α, the tree that minimizes $C_\alpha(T)$. In practical applications, hyper-parameters such as α are typically chosen by cross validation and/or grid search when more than one hyper-parameter is involved.

2.2 Ensemble learning methods

Ensemble learning is a subset of machine learning methods that is used to enhance the performance of a machine learning model combining several models (learners). Ensemble learning methods consist of two primary sub groups; Sequential Ensemble (Boosting) and Parallel Ensemble (Bagging). In the practical application part of this paper we will implement Random Forest (a form of Parallel Ensemble performed on Decision Trees), and Gradient Boosted Trees (a form of Sequential Ensemble performed on Decision Trees).

2.2.1 Parallel Ensemble (Bagging)

Bootstrap aggregation, or *Bagging*, is a form of *parallel ensemble* where one use bootstrapping to improve estimates (predictions) by *reducing variance*.

Bagging works by averaging the prediction of an input x over a collection of models trained on different bootstrap samples, thereby reducing the estimates variance.
The bagging estimate is defined as

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x)$$

where $\hat{f}^{*b}(x)$ is the predicted value for x for each model $b = 1, 2, \ldots, B$ trained on the bootstrap samples Z^{*b}, $b = 1, 2, \ldots, B$. The bagged estimate will only differ from the estimate attained by the model trained on the entire training set Z when the model function is a nonlinear or adaptive function of the data (Hastie et. al., page 282[10]).

Further, for the case of regression and squared loss, bagging can be shown to always improve upon, or attain an equivalent expected error rate as the model trained on the entire training set (Hastie et. al., page 285, equation 8.52[10]). Due to nonadditivity this doesn't hold in general for classification under $0-1$ loss, where bagging a good classifier can make it better, and bagging a bad classifier can make it worse.

Bagging Decision Trees
Decision Trees are good candidates for bagging, as they typically have high variance (i.e. splitting training data into two parts at random and fitting models on both halves often can lead to quite different models). Moreover, as each tree generated in bagging is identically distributed, the expectation of an average of B such trees is the same as the expectation of any one of them. However, when we use bagging on a large number of decision trees, it's no longer possible to represent the model using a single visual decision tree, and some interpretability is lost.

2.2.2 Random Forest

Random Forest improve upon bagged decision trees with a small change that decorrelates the decision trees. Rather than considering the full set of p predictors when making the splits, a random sample of m predictors is chosen as candidates for each splitting point. This helps decorrelating the trees by not having dominating predictors be the top splitting-variables for most of the trees — and thus in turn making the final predictions less correlated. We used the following algorithm to implement Random Forest in our practical application:

Random Forest Algorithm

1. For $b = 1$ to B:

(a) Draw a bootstrap sample $Z*$ of size N from the training data

(b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the maximum node size n_{min} is reached

i. Select m variables at random from the p variables

ii. Pick the best variable/split-point among the m selected variables using the CART algorithm described in the sections on decision trees

iii. Split the node into daughter nodes

2. Output the ensemble of trees $\{T_b\}_1^N$

To make a prediction at a new point x: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
Typically the hyper-parameter m is set to be $m \approx \sqrt{p}$.

2.2.3 Sequential Ensemble (Boosting)

Additive modeling
Generally, additive functions can be decomposed into the addition of M sub-functions:

$$F_M(x) = f_1(x) + \cdots + f_M(x) = \sum_{m=1}^{M} f_m(x)$$

Sequential Ensemble, or *boosting*, is a form of additive modeling where we sequentially apply a weak learning algorithm to repeatedly modified versions of training data, emphasizing the observations with the largest prediction error. We thereby produce a sequence of weak models (learners) $f_m(x), m = 1, 2, \ldots, M$. To simplify the analysis, we will here discuss *boosting* in the context of *regression trees* and squared error loss. It can be shown that squared error loss is far less robust than some other loss functions for this purpose (Hastie et. al, page 349[10]), but due to computational constraints it is still one of the most popular cost functions for regression trees.

In general, additive modeling can often be done by building the individual $f_m(x)$ terms in parallel and independent of each other, however this is not the case for boosting. Rather, boosting constructs and adds weak models (learners) in a stagewise and greedy fashion, where choosing $f_m(x)$ never alters the previous functions f_{m-i}, $i = 1, \ldots, m-1$.

Thus, one could choose to stop adding new models when $F_M(x)$'s performance is good enough, or when further alterations doesn't improve the model. In real life applications M is chosen as a hyper-parameter.

2.2.4 Gradient Boosting Machines

We chose the following algorithm for our implementation of Gradient Boosting Machines (Boosted Trees):

1. Let $F_0(x) = \frac{1}{N}\sum_{i=1}^{N} y_i$

2. For $m = 1, 2, \ldots, M$:

(a) Let $r_{m-1} = y - F_{m-1}(x)$ be the residual direction vector

(b) Train regression tree δ_m on r_{m-1}, minimizing squared error

(c) Set $F_m(x) = F_{m-1}(x) + \eta\delta_m(x)$, where η is a hyper-parameter often called the learning rate

For our practical application we optimized the multiple hyper-parameters through gridsearch and cross validation.

Gradient boosting machines perform optimization techniques from numerical methods called gradient or steepest descent. For a more elaborate discussion, see e.g. explained.ai's article[16].

2.2.5 XGBoost

XGBoost have many advantages over the standard GBM implementation[2][13]:

- Built in regularization — lasso and ridge (for a discussion regularization, see our first project[3])
- Parallel processing and implementation on *Hadoop* for increased speed
- High flexibility in terms of custom optimization objectives and evaluation criteria
- Built in routine to handle missing data
- A less greedy algorithm for pruning trees, leading to better model fits
- Built in Cross Validation at each iteration

To compare our implementations against the "state of the art" in machine learning, XGBoost was also included in our practical application. We should however note that getting the best possible results

from all algorithms, and perhaps especially XGBoost requires a lot of parameter tuning — thus the comparison might not be entirely apples and apples.

3 Practical application

3.1 Implementation of algorithms

All self-implemented algorithms (Decision Trees, Random Forests, Gradient Boosting Machines) are implemented according to the theory-section in the paper. Further, all source code and an excerpt of the most relevant results are listed on our project github[5].

Our implementation uses variance as the measure of impurity. As speed is important when implementing decision trees due to the large number of calls, we used the form "mean of square - square of mean" of variance in order to be able to make incremental updates to our measure of impurity as we loop through the samples looking for the best splitting threshold. This enables us to avoid an $O(n^2)$ implementation. Despite this we still have a significantly slower implementation than that of sklearn, and will thus turn to the sklearn implementations for some of this analysis in order to save time, and take advantage of some of the features from their implementations that we have not implemented. A more detailed comparison of our implementations to that of sklearn can be found in our github repo in the *analysis notebook*. We see from our analysis that our decision tree implementation matches that of sklearn, and that our implementation of random forest and GBM mathces in terms of generalization ability. However, the speed of our implementation is much slower compared to that of sklearn.

3.2 Description of data set and studied problem

For our practical application we chose the *Wine Quality Data Set*, available through UCI's Machine Learning Repository[6]. The reference contains two data sets, one for predicting the quality of red wine, and one for predicting the quality of white wine. Both wine types are variants of the Portuguese "Vinho Verde" wine. We chose to focus on the set containing data on red wine, which includes a total of 1599 total instances.

Output variable
The output variable is perceived wine quality, and ranges between 0 (very bad), and 10 (very excellent), where each sample observation is based upon at least 3 evaluations made by wine experts.

Input variables and link to focus areas in the theory section
The data set contains 11 input variables based on physicochemical tests:

- Fixed acidity
- Volatile acidity
- Citric acid
- Residual sugar
- Chlorides
- Free sulfur dioxide
- Total sulfur dioxide
- Density
- pH
- Sulfates
- Alcohol

All the input variables are numeric. We thus did not discuss/analyze specific considerations that need to be addressed when working with categorical inputs for e.g. decision trees (for a discussion, see e.g. Hastie et. al page 310[10]). Similarly, the data set didn't have any missing predictor values — discussions pertaining to e.g. decision trees advantageous handling of missing predictor values is thus left out (see e.g. Hastie et. al page 311[10]).

Commentary on relevance to past research

From our findings, past research on this data set focuses white wine or classifications tasks – see e.g. Cortez et. al, 2009[7]. Direct comparison of performance metrics is thus hard — we thus focus here on comparing the different algorithms we apply to the data against each other, rather than make comparisons across different scientific papers.

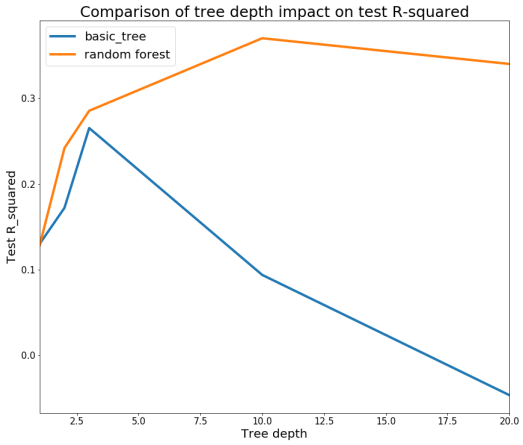
3.3 Analysis and results

For our analysis we split the data set into two parts, one for training and one for testing. We use a 60/40 split, as our data set is of limited size and we want our results to be robust. We are interested in comparing the algorithms in terms their ability to generalize to the test set. We also provide some insights into the topic of wine based on our results.

3.3.1 Benefits of bagging over normal Decision Trees

As stated in the theory section, prediction accuracy benefits a lot from ensembling multiple decision trees into a single predictor. To make that clear, we ran an experiment where we fit an increasingly deep tree to the data, and measured the test set performance of that one compared to a random forest using the same tree depth.

Figure1: Comparison of tree depth impact on test R^2

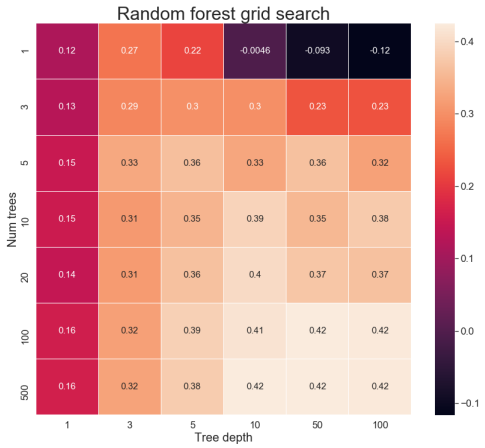


As we can see from *Figure*1, the test set performance of our decision tree peaks at a depth of 3 and quickly drops of due to overfitting. However, the random forest are able to improve for much longer, and performs well even at a depth of 20. This highlights the benefits of bagging, and how the averaging of many fairly uncorrelated trees (unrelated due to a random subset of features at each split) will take advantage of the central limit theorem to stabilize the final estimate.

3.3.2 Performance effects of optimizing hyper-parameters: Random Forests and GBMs

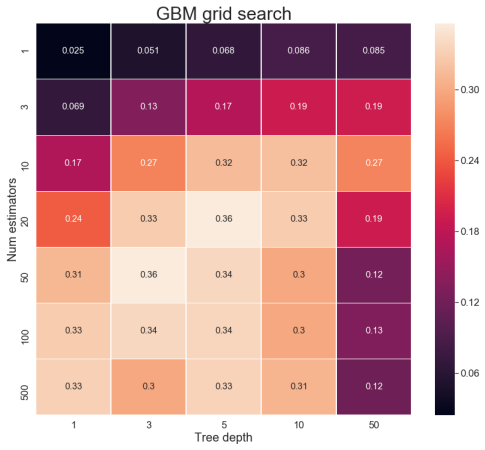
To examine the effect of the hyper-parameters on model performance, we do a grid search in order to look at how different configurations affects performance. The performance is measured by the test set R-squared. We start by looking into the plot for random forest, where we look at the two most important hyper-parameters, namely the number of trees and the depth of each tree.

Figure2: Optimizing hyper-parameters for Random Forest, effect on test R^2



We can see a clear trend in this picture, which is in line with what we would expect. The number of trees parameter is not benefiting us as much for shallow trees, as we are not able to take full advantage the diversity and independent predictions we get from the deeper trees. We see a similar picture for small number of deep trees. Here, they seem to overfit and does not perform well. We see the best performance where are able to fit a large number of deep trees. Here we are able to take full advantage of the variability in the predictions and the averaging effect.

Figure3: Optimizing hyper-parameters for GBMs, effect on test R^2



4 Conclusion

One of the major problems with trees is their high variance — a small change in the training data can often result in a very different series of splits. This is mitigated by the use of *bagging*, but at the cost

of losing the simple, tree-based model structure.

The main idea of decision trees is to find those descriptive features which contain the most information regarding the target feature and then split the dataset along the values of these features such that the target feature values for the resulting underlying datasets are as pure as possible.

Pros and cons of trees, pros White box, easy to interpret model. Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches discussed earlier (think of support vector machines) Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression! No feature normalization needed Tree models can handle both continuous and categorical data (Classification and Regression Trees) Can model nonlinear relationships Can model interactions between the different descriptive features Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small)

Disadvantages Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches If continuous features are used the tree may become quite large and hence less interpretable Decision trees are prone to overfit the training data and hence do not well generalize the data if no stopping criteria or improvements like pruning, boosting or bagging are implemented Small changes in the data may lead to a completely different tree. This issue can be addressed by using ensemble methods like bagging, boosting or random forests Unbalanced datasets where some target feature values occur much more frequently than others may lead to biased trees since the frequently occurring feature values are preferred over the less frequently occurring ones. If the number of features is relatively large (high dimensional) and the number of instances is relatively low, the tree might overfit the data Features with many levels may be preferred over features with less levels since for them it is more easy to sp

5 Appendix

References

[1] Lessons from 2mm machine learning models, import.io.

[2] Xgboost: A scalable tree boosting system, author =.

[3] Johannes A. Barstad and Olve Heitmann. Project1 github repository. https://github.com/Barstad/fys_stk3155/tree/master/PROJECT1.

[4] Johannes A. Barstad and Olve Heitmann. Project2 github repository. https://github.com/Barstad/fys_stk3155/tree/master/PROJECT2.

[5] Johannes A. Barstad and Olve Heitmann. Project3 github repository. https://github.com/Barstad/fys_stk3155/tree/master/PROJECT3.

[6] Cerdeira A. Almeida F. Matos Cortez, P. and J. T., Reis. Wine quality data set. https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

[7] Cerdeira A. Almeida F. Matos Cortez, P. and J. T., Reis. Modeling wine preferences from physicochemical properties using fuzzy techniques. https://www.scitepress.org/Papers/2015/55519/55519.pdf, 2009.

[8] Norway Department of Physics, University of Oslo. Project 3 assignment description. https://compphysics.github.io/MachineLearning/doc/Projects/2019/Project3/pdf/Project3.pdf.

[9] Cernadas E. Barro S. Fernández-Delgado, M. Do we need hundreds of classifiers to solve real world classification problems? http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf, 2014.

[10] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, 2008.

[11] M. Hjorth-Jensen. Data analysis and machine learning: From decision trees to forests and all that. https://compphysics.github.io/MachineLearning/doc/pub/DecisionTrees/html/DecisionTrees.html, 2019.

[12] Morten Hjorth-Jensen. Data analysis and machine learning lectures: Optimization and gradient methods. https://compphysics.github.io/MachineLearning/doc/pub/Splines/html/._Splines-bs000.html.

[13] A. Jain. Complete guide to parameter tuning in xgboost with codes in python, 2016.

[14] La Cava W. Mustahsan Z. Varik A. Olson, R.S. and J.H. Moore. Data-driven advice for applying machine learning to bioinformatics problems, 2018.

[15] Howard J. Parr, T. Gradient boosting: Distance to target. https://explained.ai/gradient-boosting/L2-loss.html.

[16] Howard J. PArr, T. Gradient boosting performs gradient descent. https://explained.ai/gradient-boosting/descent.html.

[17] USGS. Uci machine learning repository - default of credit card clients data set. https://earthexplorer.usgs.gov/.

[18] Tivon Von Vivo. Artificial vintelligence: Ai gets taste of wine industry. https://vonvino.com/artificial-intelligence/, 2018.