Detecting Manipulated Wine Ratings with Autoencoders and Supervised Machine Learning Techniques

Michaela Baumann
Business Intelligence / Analytics Competence Center
NÜRNBERGER Versicherung
Nürnberg, Germany
email: michaela.baumann@nuernberger.de
ORCID: 0000-0001-5066-9624

Michael Heinrich Baumann
Department of Mathematics
University of Bayreuth
Bayreuth, Germany
email: michael.baumann@uni-bayreuth.de
ORCID: 0000-0003-2840-7286

***Abstract*—In this study, we analyze the ability of different machine learning methods to detect manipulated wine ratings. We consider autoencoders, regression models (neural networks, support vector machines, random forests) and classification models (support vector machines, random forests) and two different kinds of manipulation strategies. We find that autoencoders perform best on unmanipulated test data, i.e., their reconstruction error is smaller than the supervised models' prediction error. However, on the manipulated test data, the supervised models outperform autoencoders. This is interesting since autoencoders are generally used for outlier detection. When comparing only the supervised methods, we find that, basically, both support vector machines and random forests perform and detect better than regression neural networks. Additionally, the optimization and training times for these two model types are smaller. In order to consider a relatively large grid of hyperparameters especially for the neural networks, we introduce a hyperparameter tuning method called sequential accumulative selection. To sum up, when trying to detect manipulations, different methods have usually both advantages and disadvantages.**

Keywords*—*anomaly detection; manipulation identification; wine preferences; artificial neural networks; autoencoders; support vector machines; random forests.

I. Introduction

The work at hand is an extended version of [1]. Some parts are identical to the conference paper [1] (May 22 version and Oct. 22 version), however, we especially modified and extended the methodology (Section IV) by considering a larger set of models and different data manipulation strategies and, therefore, get new and more detailed results (Section VI).

In a world of increasingly differentiated products and customers who frequently change their buying behavior, it is difficult to assess whether the price-performance ratio is appropriate before making a purchase. An important and much-used assistance in such buying decisions are ratings. In this study, we are going to approach the question of whether and how manipulated ratings can be detected using wine quality ratings as an example. When ratings come from an official or non-official authority (such as Gambero Rosso's *Vini d'Italia* [2], Robert Parker's *The Wine Advocate* [3], *Gault&Millau* [4], or *Guide Michelin* [5], when dealing with wines, hotels, restaurants, or related topics), it is possible to verify with little effort whether ratings given by a merchant or producer are genuine by simply looking up the relevant work. However, since by far not all wines are represented and rated in one of the works published by an authority, there are countless other ratings. These other ratings, which are not given by an authority, are difficult to verify for authenticity, and it might even be possible that they are not objective, but rather paid for by someone. In the following, we are going to show possibilities for detecting such manipulated or faked ratings.

A very basic idea for how to identify manipulated ratings would be via (linear) regressions (cf. [6]). That means, when we have other, exactly measurable features, such as alcohol content, pH value, or density, we can learn how to predict the rating using these independent variables on correctly rated data objects. Ratings that differ (strongly) from the predicted ones on unseen data might be suspicious. However, a linear regression does not lead to good results in our case, i.e., when trying to detect manipulated wine ratings. Thus, the research question is how manipulated wine ratings may be detected in a better way.

Since artificial neural networks are currently en vogue, one can of course use a regression by means of a neural network (cf. [7]–[10]). Note that a linear regression is the same as an exactly trained, fully connected neural network without any hidden layer with linear activation functions (i.e., id resp. pass-through), when adding a dummy column (filled with 1s) in the data for the intercept and using *Mean Squared Error* (MSE) as loss. Regressions based on shallow or deep neural networks are likely to outperform a linear regression. Other machine learning techniques for performing regression tasks are, for example, support vector machines [11][12] or random forests [13][14].

Especially when dealing with outlier detection, so-called autoencoders (resp. reconstruction networks or autoassociative neural networks) are a common means [15]–[18]. Autoencoders consist of two parts (i.e., two regressions), an encoder and a decoder. The encoder compresses the input data to a lower dimensional representation usually referred to as the code; the decoder takes the code as input and aims to reconstruct the original input.

Given a well trained autoencoder, when the input and the output differ (strongly), the data might be manipulated (or in other contexts: an outlier, an anomaly, fraudulent, or suspicious). Note that there are much more application areas of autoencoders, such as dimensionality reduction, data compression, or denoising. Although it is in principal assumed that the quality depends on the other features, the autoencoder does not use this dependence information, that is, the quality and all other features are considered as coequal input (and output) variables. Since the autoencoder does not use all the information that is actually available, it would be very interesting if it nevertheless achieved better results.

In the work at hand, we investigate how *Regression Neural Networks* (RNNs), *Support Vector Machines* (SVMs), *Random Forests* (RFs) and *Neural Network based Autoencoders* (NNAs) can be used to identify manipulated data. Additionally, as benchmark models we use a linear regression (*Linear Model;* LM; see [6]) and an autoencoder that implements two linear regressions (*Benchmark Autoencoder;* BA; see Section IV-D).

In the work at hand, we summarize RNN, SVM, RF and LM under the term *supervised methods,* since these models are trained with an explicit label, the wine quality. The autoencoders NNA and BA are not referred to as supervised methods. Usually, autoencoders fall into the category of unsupervised methods since there is no explicit target feature or label, but only covariates. This reflects the usage of autoencoders when the code layer is of interest, e.g., when conducting dimensionality reduction or data compression. Even though the training process does not differ, when the reconstructed input explicitly is of interest, autoencoders may in this case be called *autoassociative* or *self-supervised methods* [19][20].

There clearly are several other data analytics methods that might be applied, especially since the number of those techniques keeps growing (see [21][22]), however, the methods used here are conceptually different from each other and therefore cover a reasonable range of possible methods.

We find that both regression and classification techniques are suitable for detecting manipulated wine ratings. Autoencoders as means for uncovering outliers in data sets do not detect the manipulated data objects as well as the supervised methods. Although we provide a relatively large grid of possible hyperparameters for the neural network based models, both RFs and SVMs show a better detection performance than the RNN and the NNA. Further, the variability in the results of the neural network based models is quite high.

The remainder of this paper is organized as follows: Section II reviews both the literature on wine data analysis and those on anomaly detection in general while Section III specifies the data we are using. Section IV depicts our methodology step by step and Section V describes the preparatory activity for the hyperparameter optimization of the neural networks called sequential accumulative selection. Section VI presents the results. Finally, Sections VII and VIII conclude and describe possibilities for ongoing work.

II. Literature Review

The closely related literature roughly splits into two groups, namely data analytics of wine quality and general outlier/anomaly detection resp. fraud identification. The analytics of wine quality mostly covers the prediction of wine ratings based on measurable features. Cortez et al. [23][24] compare several data mining regression methods for predicting wine preferences based on easily available data during the certification of wines. In this context, they originally published the two datasets that are also used in the work at hand. They use the vinho verde white wine dataset [24] and both the vinho verde red wine and white wine datasets [23]. The data mining methods for predicting wine quality in both papers are neural networks, i.e., multilayer perceptrons, and SVMs. Besides these papers, also the importance of the selection of the most relevant features before predicting the wine quality with machine learning regression methods is investigated for the vinho verde datasets [25]. Here, a linear regression is used for assessing the importance of the variables. Neural networks and SVMs are then trained for the actual prediction of the wine quality via regressions. The neural network and SVM results are compared when using all available variables or only the most important ones as assessed by the linear regression. The vinho verde white wine dataset is used for classifying wine preferences via fuzzy inductive reasoning [26]. To increase statistical confidence, in [26], the evaluation process is repeated 20 times. Deep neural regression networks are applied for predicting wine ratings on the vinho verde datasets [27]. In [27], the authors train the neural networks separately for the red and white wine datasets and find that different network architectures are needed for the two datasets. These results are then compared to a multiclass SVM, where the neural networks outperform the SVM.

An expert model consisting of several submodels for different types of input variables for assessing wine quality is developed to assist the winemakers in their business [28]. This model is evaluated on 45 data samples from southern France consisting of altogether 137 variables (vineyard variables and enological variables). In other works, the effect of weather and climate changes as well as the effect of expert ratings on the prices of Bordeaux wines are analyzed [29][30]. With this, the efficiency of the Bordeaux wine market is assessed. Tree models are used for predicting the relative quality of German Rhinegau Riesling considering terrain characteristics obtained through cartographic studies [31]. Due to unbalances in the original seven target categories, the Riesling quality classification of [31] is developed for a target variable mapped from the original seven to three more equally balanced categories. In another work, a framework is developed that automatically finds an appropriate set of classifiers and hyperparameters via evolutionary optimization for predicting wine quality for arbitrary wine datasets [32]. That work also gives a good overview over further research concerning the prediction of wine quality with several machine learning methods, such as k-nearest neighbors [33][34], naive Bayes [35][36], or RFs [34][36].

Having in mind the literature briefly reviewed above, which predicts wine ratings or conducts data analyses of wine quality, the work at hand contributes by connecting wine rating predictions and anomaly detection. The topic of outlier/anomaly detection and fraud identification is addressed in a lot of related work in various contexts (see, for example, the surveys and summaries [37]–[41]) and we can only touch on this broad topic here. Generally, according to Chandola et al., "*Anomaly detection* refers to the problem of finding patterns in data that do not conform to expected behavior" [37]. Usually, anomalies have to be identified throughout the analysis of data so that they can be treated separately and do not distort the results of the analysis of "normal" data. However, in the case of fraud and also in our case of manipulation detection they are of special interest (cf. [37]). Fraudulent and manipulated data objects inhibit abnormal patterns but they try to appear as normal. The detection of anomalies, especially of intentional, malicious anomalies, such as fraud or manipulation, is very challenging and there are many approaches that try to accomplish this task. The approaches basically fall in one of the following three categories [38]:

- Unsupervised methods (e.g., clustering); labels are not needed here and new patterns (normal ones and outliers) may be processed correctly.
- Supervised methods (e.g., classification); these need prelabeled data, however, anomalies are usually very rare and the labeled datasets are, thus, highly unbalanced; new patterns are unlikely to be processed correctly.
- Semi-supervised methods; normal behavior is known, i.e., (a part of) the training data is labeled as normal, and new, unlabeled data objects are compared to the normal case.

When we apply the models mentioned in Section I for detecting manipulated wine ratings, we use them in a semi-supervised fashion. This means, the supervised models RNN, SVM, and RF are trained for predicting the target feature "wine quality" and the NNA is trained for reconstructing all features (autoassociative resp. self-supervised). In each case, this training is carried out on unmanipulated wine data. Whether the wine data is manipulated or not is a second label/target feature that is not present in the training process or, in other words, that is the same (namely: not manipulated) for the whole training data set. For assessing the manipulation detection ability of the models, the supervised models' predictions and the autoencoder's reconstructed features of the test data, which reflect the normal, i.e., unmanipulated case, are compared to the unlabeled (in terms of manipulation) test data. The models themselves cannot be semi-supervised, but the detection approach as a whole is semi-supervised.

In addition to methods that require tabular data (a priori tabular data, but also image, audio, or video data transferred to tabular data) there are methods that operate on graph based data [42], which are especially useful when identifying anomalies in highly connected data. The approach of the work at hand falls into the third category, i.e., semi-supervised methods, and works on tabular data.

III. Data

The approach described in this work is applicable to various working areas (see Section VIII). We demonstrate it using wine data as an example because of the following reasons.

A rather simple advantage is the good data availability and (if no wine names or winemaker names are used) the innocuousness of the data. Further, the explaining variables (except for wine or winemaker names) are metric, clearly defined, and exactly measurable (e.g., alcohol content, acid, pH value, red/white). The used datasets further have a unique target feature and not a list of ratings (see also Section VIII).

We use the "Wine Quality Datasets" [23] from the Universidade do Minho [43], more specifically the datasets "White Wine Quality—Simple and clean practice dataset for regression or classification modelling" [44] and "Red Wine Quality—Simple and clean practice dataset for regression or classification modelling" [45] downloaded from *kaggle,* which are licensed under "Database Contents License (DbCL) v1.0," *Database: Open Database, Contents: Database Contents* [46]. Both datasets contain anonymized *vinho verde* wines and have the same twelve columns: eleven independent variables summarized in Figure 1 through boxplots and the dependent variable summarized in Figure 2 by means of a histogram. The dependent variable *quality* is the wine rating, which is supposed to depend on the other, explaining features, called independent. All values except for the ratings are in some meaningful physical unit, while the ratings range from 0 (very bad) to 10 (excellent) in integer steps [23]. The red wine dataset consists of 1,599 entries while the white wine data has 4,898 rows, leading to a combined data set with 6,497 rows and 13 columns.

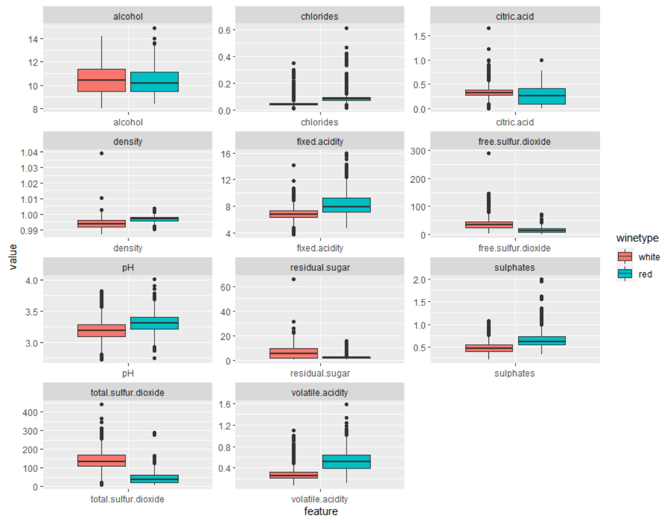

Figure 1. Summary of the distribution of the independent features for the white and red wines via boxplots

For the distinction of red and white wines we added a binary encoded categorical column to the union of both datasets. There already exist extensive analyses of the vinho verde datasets covering, among others, correlations, clusterings, distribution estimations, etc. Such statistics and many more analyses can be found in the work of Cortez et al. [23][24], in other papers [25]–[27], and further tutorials or notebooks [47]–[51].

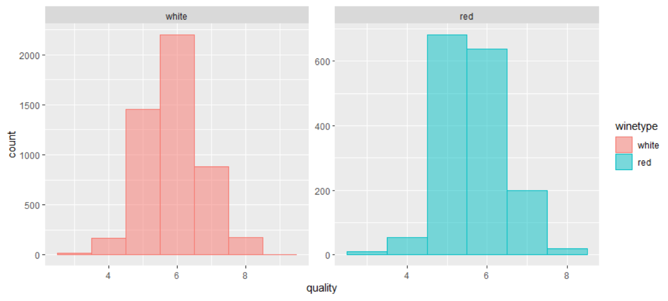

Figure 2. Summary of the distribution of the dependent variable for the white and red wines via histograms

IV. Methodology

As outlined in Section I, the aim of this work is to identify manipulated ratings. For this, we train several network based models, SVMs, and RFs on provided, correct data. We then make predictions on unseen data objects where we manipulate a certain part of these objects. We use two different kinds of manipulation strategies. In the first one, we increase the original rating of very low rated wines as this seems to be a "reasonable" manipulation in the context of wine ratings, e.g., when someone wants to increase sales numbers. In the second one, we take a random part of the test data and replace the wine rating by a value drawn randomly from the empirical distribution of the remaining, unmodified part of the test data. By comparing the potentially manipulated data and the predicted data we aim at identifying the manipulated data objects. We assume that objects for which the predicted values strongly differ from the provided data are more likely to be manipulated. We assess the models' detection performance, i.e., their ability to identify manipulations through calculating the true and false positive rates when marking the most deviating data objects as suspicious. To prevent overfitting and account for other random effects we apply bootstrapping. That is, we repeat the process of randomly splitting the data and training the models such as, for example, also done in [26]. Finally, we take among others the median over the particular results.

In the following, we explain our methodology in detail. The implementation is done in `R` using the `Keras` library, which is an API to `TensorFlow`, for the neural networks, the `e1071` library for the SVMs, and the `caret` library with the `rf` option for the RFs.

A. Bootstrapping and Data Splitting

The bootstrapping is in our case a Monte Carlo like approach of repeatedly and independently splitting the complete dataset $\underline{all}$ (6,497 rows, 13 columns) with a ratio of 70:30 into training data (4,547 rows, 13 columns) and test data (1950 rows, 13 columns) 100 times: $\underline{all} = \underline{train} \mathbin{\dot{\cup}} \underline{test}$. To make this process reproducible, we set an initial seed and randomly draw 100 seeds $(seed_1, \ldots, seed_{100})$. Before every splitting we explicitly set the seed to the respective run's seed. Please note that when conducting the hyperparameter optimization for the different methods, the training data set $\underline{train}$ is split automatically inside the respective `R` functions into a development and validation data set or into the different folds when using a k-fold cross validation.

B. Data Manipulation

We use and compare two wine rating manipulation strategies. In the first one, we manipulate the 10% worst ranked test data by averaging the original rating and the highest possible rating (10) and rounding up. That is, we split the test data $\underline{test} = \underline{low} \mathbin{\dot{\cup}} \underline{high}$ with a ratio of 10:90 (with a random tie breaking), manipulate $\underline{low} \mapsto \underline{low_{manip\_worst}}$ and get the manipulated test data

$$\underline{manip_{worst}} := \underline{low_{manip\_worst}} \mathbin{\dot{\cup}} \underline{high}.$$

We also add a flag column to the manipulated test data for marking the manipulated entries for evaluation purposes.

In the second manipulation strategy, we randomly take 10% of the test data and replace the true rating by a rating value drawn from the empirical distribution of the remaining, unmodified part of the test data. That is, we split the test data $\underline{test} = \underline{random} \mathbin{\dot{\cup}} \underline{rest}$ with a ratio of 10:90 (again with a random tie breaking). The manipulation replaces $\underline{random} \mapsto \underline{random_{manip\_random}}$ such that we get the manipulated test data

$$\underline{manip_{random}} := \underline{random_{manip\_random}} \mathbin{\dot{\cup}} \underline{rest}.$$

Of course, in the second manipulation strategy it may happen for some or even for all wines, that the true rating and the manipulated rating are the same due to the random drawing of manipulated ratings. That is, we actually have a rating manipulation of at most 10% randomly drawn wines. Further, the changes of the true and the manipulated ratings may be rather small, especially compared to the changes induced by the first manipulation strategy. This is why we expect all detection methods to "perform," i.e., to detect better on the test set manipulated with strategy one, i.e., on $\underline{manip_{worst}}$, than with strategy two, i.e., on $\underline{manip_{random}}$.

C. Data Normalization

As can be seen in Figure 1, the scales of the independent wine features differ strongly. Most of the machine learning models have problems with differing scales, which is why some data preprocessing, in our case the normalization of the data, is necessary. We normalize the independent features by min-max-scaling where $\underline{train}$ serves as reference. That means, also the test datasets are normalized with the minimum and maximum values of $\underline{train}$. For LM, RNN, SVM, and RF the target variable "quality" is not normalized. For BA and NNA, "quality" is an input variable like the others and, hence, normalized. To obtain comparable results, the performance of the regression models is normalized afterwards (using $\underline{train}$).

D. Models

We consider six different kinds of models: LM, RNN, SVM, RF, BA, and NNA. Except for the two autoencoders (BA and NNA), the other model types also appear in the related work concerning the prediction of wine quality, see Section II. The two simple models LM and BA are solely for benchmarking the general performance of the other models on the unmanipulated test data $\underline{test}$.

Because SVMs and RFs are made primarily for classification, we optimize and train SVM and RF not only as regression models, but also as classification models. This means, for regression, we treat the wine quality as a numeric variable with values in $[0, 10]$, for classification we treat it as a factor variable with values in $\{0, 1, \dots, 10\}$. We refer to the classification models as SVMclass and RFclass, the regression models are further named SVM and RF.

Thus, we measure the manipulation detection performance for RNN, SVM, SVMclass, RF, RFclass, and NNA. The particular model configurations are described in Section IV-E. For the two simple models, the following holds: LM uses `R`'s `lm` function. BA is a fully connected, three layer network with input layer (size 14), code layer (size 4), and output layer (size 14). The input is the 13 dimensional data plus a constant column of 1s (intercept) in order to mimic two nested linear regressions. This is also the reason why linear activation functions and MSE as validation metric are used.

E. Hyperparameter Tuning

In every step during the bootstrapping, all models except the two simple ones are trained with hyperparameter optimization over a grid. Especially for the neural network based models RNN and NNA, the respective grids are quite large (see also, e.g., [27]). We apply the so-called sequential accumulative selection, see Section V, in a preceding step in order to possibly reduce the grid size for the actual tuning and training. The tuning of the neural network based models uses simple splits into development and validation data due to runtime issues. That means, we do not apply a k-fold cross validation here (at least not for $k > 1$). The hyperparameter grid after applying the sequential accumulative selection for RNN is:

- Activation function (hidden layers): `linear`, `softplus`, `ReLU`
- Activation function (output layer): `linear`
- Number of hidden layers: 1, 3, 5, 7
- Dropout rate: 0%, 5%, 10%
- Number of neurons in each hidden layer: 32, 64, 128
- Number of neurons the input layer: 12
- Number of neurons the output layer: 1
- Batch size: 32, 64
- Learning rate: 5%, 10%
- Patience for early stopping: 15
- Patience for learning rate reduction: 7
- Loss function: MSE
- Evaluation measure: *Mean Absolute Error* (MAE)
- Optimizer: `Adam`
- Number of epochs: 75
- Batch normalization: between every layer

The grid for NNA after the sequential accumulative selection is defined as follows:

- Activation function (hidden layers, except the code): `softplus`, `ReLU`
- Activation function (code layer and output layer): `linear`
- Number of hidden layers (except code layer): 4, 6
- Dropout rate: 0%
- Number of neurons in each hidden layer (except the code): 64, 128
- Number of neurons the input layer as well as in the output layer: 13
- Number of neurons the code layer: 4
- Batch size: 32, 64
- Learning rate: 5%, 10%
- Patience for early stopping: 15
- Patience for learning rate reduction: 7
- Loss function: MSE
- Evaluation measure: MAE
- Optimizer: `Adam`
- Number of epochs: 75
- Batch normalization: between every layer

For training the SVMs (SVM and SVMclass), we perform a five-fold cross validation with hyperparameter tuning over the following grid:

- Kernel: `linear`, `radial`
- Gamma: 0.01, 0.1, 1
- Cost: 0.01, 1, 100

As regression mode of SVM we set the ε-regression with the `e1071` package default for ε. As classification mode for SVMclass we set the standard C-classification. We use the same grids for SVM and SVMclass.

The two models RF and RFclass are also trained using a five-fold cross validation with hyperparameter tuning. The hyperparameter grids are, just as above, the same for regression and classification:

- Number of variables considered for splits (.mtry): 1, 2, 3, 4, 5, 6
- Number of trees: 500, 1000

For the mtry parameter, there exists the rule of thumb to use $\lfloor \frac{p}{3} \rfloor$ for regression tasks and $\lfloor \sqrt{p} \rfloor$ for classification tasks with p being the number of explaining variables [52]. This is why we vary mtry around $\lfloor \frac{12}{3} \rfloor = 4$ resp. $\lfloor \sqrt{12} \rfloor = 3$.

F. The Algorithm

The bootstrapping, model training, and evaluation algorithm is depicted in the algorithm in Figure 3. All individual steps are described above. The algorithm is suitable for parallelization as the bootstrapping runs, i.e., the steps executed within the for-loop that begins in line 2 of Figure 3, are completely independent of each other.

In the hyperparameter optimization step (line 6) we use MSE as performance measure for SVM, the *Root Mean Squared Error* (RMSE) for RF, accuracy for RFclass and

```
begin
for i=1 to $n$ do
    begin
    Prepare datasets with $seed_i$ (split, manipulate, normalize);
    Train the two benchmark models on $\underline{train}$;
    Optimize hyperparameters of RNN, SVM, SVMclass, RF, RFclass, and NNA and retrain the best model in each case on $\underline{train}$;
    Measure all models' performance on $\underline{test}$;
    Measure detection performance of RNN, SVM, SVMclass, RF, RFclass, and NNA on $\underline{manip_{worst}}$ and on $\underline{manip_{random}}$;
    end
end
```

Figure 3. Procedure for model training and evaluation. Input: the original dataset; a seed vector $(seed_1, \ldots, seed_{100})$; four hyperparameter grids. Output: list of performance data.

the misclassification error (1-accuracy) for SVMclass. In the neural networks, the loss function is MSE, however, as performance metric for the hyperparameter optimization we use MAE. For RNN, the MAE is calculated on the target variable, for NNA, the MAE is calculated over all features. Note that these different performance measures are used for optimizing the hyperparameters but not for depicting the performance in Section VI-A.

As one can see from the algorithm, our approach is supervised when we train the models for predicting wine quality (LM, RNN, SVM, SVMclass, RF, and RFclass) resp. all features (BA, NNA). However, concerning a prediction of the manipulation label, our approach may be called semi-supervised. This is because we use labeled data to train the models, but only data that is labeled as "correct," i.e., that is not manipulated. Although in the analysis "correct" and "incorrect," i.e., manipulated data are used, no incorrect data are used for training—that is, one does not need a data set where "incorrect" data are already identified as incorrect. We use the information about which data entries are really "incorrect" only for the statistical analysis of the results for this paper.

G. Detection Performance

The detection performance is measured as follows: For RNN, SVM, SVMclass, RF, RFclass, we calculate the squared difference of the predicted quality and the given quality (which is possibly manipulated) for each data object (*Squared Error;* SE). Note that the results of the two classification models are treated as numeric values here. For NNA, we compute the detection performance in two different ways: on the one hand according to the regression models via the squared differences only on the target variable and on the other hand via the sum over the squared differences of all features (*Sum of Squared Errors;* SSE). In the following, when we distinguish between these two measurement methods, we denote with NNA the performance measure on only the target variable and with NNA_all the performance measure on all variables. Note that NNA and NNA_all are not two different kinds of models (unlike RF and RFclass resp. SVM and SVMclass) but denote only the two different kinds of detection performance measurement for the same model. For each model resp. each measurement type, we sort the data in descending order according to the respective deviation values. For example, for the first manipulation strategy, we map the manipulated test data to the following resorted sets:

$$\begin{aligned} \underline{manip_{worst}} \mapsto\ (&\underline{manip_{worst,RNN}},\\ &\underline{manip_{worst,NNA}},\\ &\underline{manip_{worst,NNA\_all}},\\ &\underline{manip_{worst,RF}},\\ &\underline{manip_{worst,RFclass}},\\ &\underline{manip_{worst,SVM}},\\ &\underline{manip_{worst,SVMclass}}) \end{aligned}$$

.

Then, we determine the true/false positive rates when marking the first $q_i\%$ of the data objects in the sorted sets $\underline{manip_{worst,x}}$ with $x \in \{$RNN, NNA, NNA_all, RF, RFclass, SVM, SVMclass$\}$ as suspicious for $q_i = i$, $i = 1, 2, \ldots, 99$. The true positive rate tpr is defined as $tpr = TP/(TP + FN) = 1 - fnr$ and the false positive rate fpr is $fpr = FP/(TN + FP) = 1 - tnr$, where TP is the number of true positives, i.e., of manipulated objects that are marked suspicious, TN is the number of true negatives, i.e., of unmanipulated objects that are not marked, and FP and FN are the respective false positives/negatives and fnr and tnr the respective rates. If one would assign the "suspicious marks" randomly with equal probabilities to $q\%$ ($q \in [0, 100]$) of the data, the expected true/false positive rates would equal q, i.e., $\mathbb{E}[tpr] = \mathbb{E}[fpr] = q$, independent of the share of real positives/negatives. The values for $q = 0$ and $q = 100$ are meaningless since in the former case no object would be marked as suspicious and in the latter case all objects would be marked as suspicious.

To summarize the results of all runs, we calculate all quartiles of tpr and fpr for every q_i, i.e., minimum, first quartile, median, third quartile, and maximum. Before presenting the results of our analysis in Section VI, we describe how the set of possible hyperparameters via the sequential accumulative selection is found.

V. Hyperparameters for Neural Networks via Sequential Accumulative Selection

Since basically the set of possible hyperparameters especially for the neural network based models is infinite, it is quite natural that the size of this set has to be reduced. In doing so, for the neural networks, we perform the hyperparameter optimization in two steps. In the first step, we start with an initial set for possible hyperparameters, i.e., with a relatively large grid on the actually infinitely large space of hyperparameters. Additionally, we make an initial guess for a plausible setting as it is typical, for example, also for several optimization algorithms. In our case this means that each hyperparameter is initially set to a plausible value (underlined). This is done based on comparisons to similar problems as well as extensive trial-and-error pre-tests. We then reduce the size of the initial, large grid such that not all possible hyperparameter combinations need to be tried in the optimization step itself, which is performed as a grid search. We call the grid reduction "sequential accumulative selection."

For our case study, the initial hyperparameter grid for RNN is:

- Activation function: `linear`, `softplus`, $\underline{\texttt{ReLU}}$, `tanh`, `sigmoid`
- Number of hidden layers: 0, 1, 3, 5, $\underline{7}$
- Dropout rate: 0%, $\underline{5\%}$, 10%
- Number of neurons in each hidden layer: 32, 64, $\underline{128}$
- Batch size: $\underline{32}$, 64
- Learning rate: 5%, $\underline{10\%}$

The initial grid for NNA is:

- Activation function: `linear`, `softplus`, $\underline{\texttt{ReLU}}$, `tanh`, `sigmoid`
- Number of hidden layers (excluding the code layer): 0, 2, 4, $\underline{6}$
- Dropout rate: 0, $\underline{0.05}$, 0.1
- Number of neurons in each hidden layers (except the code layer): 32, 64, $\underline{128}$
- Batch size: $\underline{32}$, 64
- Learning rate: 0.05, $\underline{0.1}$

All other parameters are fixed to the values of Section IV-E. Note that we intentionally did not include varying numbers of neurons for the code layer of the autoencoder. This is because higher numbers of neurons in the code layer lead to a higher performance, but to a lower compression. Since both values are important for outlier detection, based on comparisons to similar examples, we chose four as a promising tradeoff.

Using the heuristic strategy of sequential accumulative selection, the two grids given above are thinned out so that the hyperparameter optimization in the algorithm in Figure 3 (in Section IV-F) performs within a reasonable runtime.

Next, we explain the sequential accumulative selection:

1) We start with performing 50 runs, i.e., on 50 different, randomly built training data sets, with the hyperparameters fixed to the underlined, plausible values except for the activation function, which is allowed to be any of the given possibilities. All activation functions that were taken at least once in the hyperparameter optimization in the 50 runs are declared to be also plausible, all others are deleted.
2) In the same fashion, the number of hidden layers is analyzed next, i.e., the hyperparameter optimizer has to optimize over the set of the plausible activation functions (due to step one there is possibly more than one plausible activation function) and the number of hidden layers. All values for the hidden layers that were chosen at least once are declared to be also plausible, all others are deleted. The plausible activation functions remain the same independent of whether they still appear in the set of optimal parameters of this second round.
3) This procedure is repeated in the following order with the number of neurons,
4) the dropout rate,
5) the batch size, and
6) the learning rate.

The results of the sequential accumulative selection, i.e., of the diminution of the possible hyperparameters, can be found in Section IV-E. For clear, the procedure of sequential accumulative selection is done separately for RNN and NNA and conducted on unmanipulated training data. The performance measure both for RNN and for NNA when selecting the best hyperparameter constellation in each run is the MAE, cf. IV-F.

Concerning the approach of the sequential accumulative selection, not only the choice of the plausible initial values for the specific hyperparameters is important, but also the order in which they are processed. Further, it is not clear in advance whether the grid is reduced at all. In the worst case, more models have to be trained than with simply using the initial, large grid. In our case, we had a distinct performance benefit through the sequential accumulative selection.

VI. Results

The results section is divided into three parts. First, we show the general model performances on the unmanipulated test data $\underline{test}$. That means, we analyze the ability of the regression models to predict correct quality values, the ability of the classification models to predict correct quality classes and the reconstruction ability of the autoencoders. Second, we examine the detection performance of the models except the benchmark models on the test data manipulated with manipulation strategy 1, i.e., on $\underline{manip_{worst}}$. Third, we investigate the detection performance under manipulation strategy 2, i.e., on $\underline{manip_{random}}$.

A. General Model Performance

The performance of all models, i.e., benchmark models LM and BA as well as the four resp. six more elaborate models SVM, SVMclass, RF, RFclass, RNN, NNA on the unmanipulated test data is depicted in Figure 4 through boxplots that capture the empirical distribution of the performance over all Monte Carlo runs. As measure for the predictive quality of all models we show the MAE where the predicted and the actual classes of the two classification models are treated as numeric values. For the two autoencoder models we show their performance on the target variable only (NNA and BA in Figure 4) and averaged over all variables (NNA_all and BA_all in Figure 4). The MAE values of the supervised models are normalized (the regression resp. classification is done on the unnormalized target feature) so that we can compare them to the autoencoder performance measurements.

As there are outliers in the performance of the neural network based models NNA and RNN, we truncated the

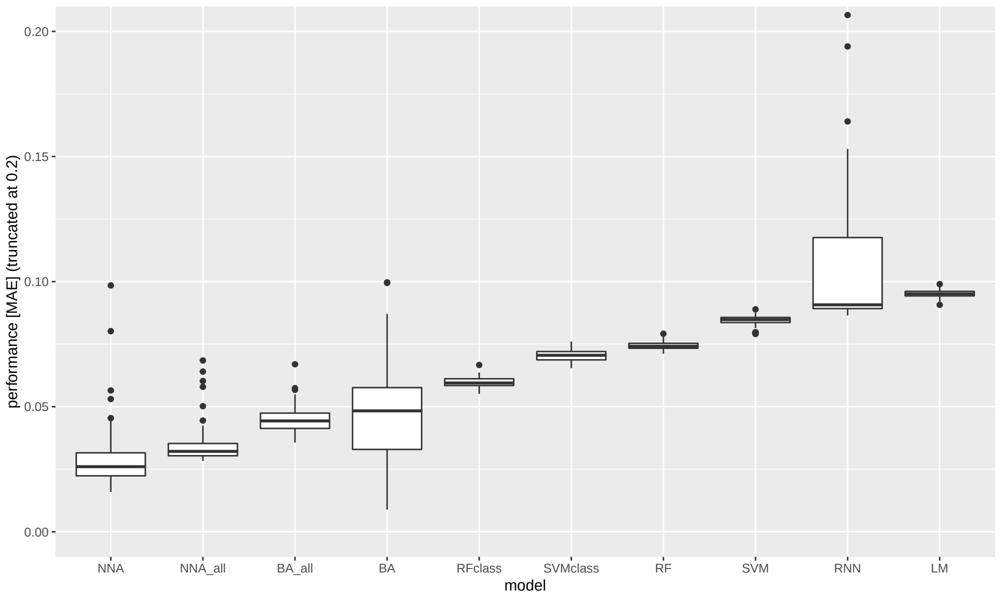

Figure 4. Boxplot of the performance (MAE) of all models (benchmark models and more elaborate models) on the unmanipulated test data. Outliers are truncated. In median, NNA is best, whilst the interquartile distance is the smallest for LM.

ordinate axis in Figure 4. The actual, rounded values of the five number summary, which is the basis for the boxplots drawn in Figure 4, is given in Table I.

TABLE I
FIVE NUMBER SUMMARY, I.E., MINIMUM, FIRST QUARTILE, MEDIAN, THIRD QUARTILE, AND MAXIMUM OF THE PREDICTIVE PERFORMANCE OF ALL MODELS.

	Min.	1st Qu.	Median	3rd Qu.	Max.
SVM	0.079	0.084	0.085	0.086	0.089
SVMclass	0.065	0.069	0.071	0.072	0.076
RF	0.071	0.073	0.074	0.075	0.079
RFclass	0.055	0.059	0.060	0.061	0.067
LM	0.091	0.094	0.095	0.096	0.099
BA	0.009	0.033	0.048	0.058	0.100
RNN	0.086	0.089	0.091	0.118	6, 106, 068.000
NNA	0.016	0.022	0.026	0.032	264.789
BA_all	0.036	0.041	0.044	0.047	0.067
NNA_all	0.028	0.030	0.032	0.035	242.428

Figure 4 and Table I show that NNA has the best predictive performance (in median) and that in general autoencoders, both NNA and BA, are better than the regression and classification models (in median). Note, however, that the application conditions for autoencoders and supervised methods are not the same. Autoencoders try to reconstruct the original input via compressing and decompressing. That particularly means that they know the value of the target variable (in terms of the supervised methods) as this is part of their input data also in the test set. It is therefore not clear whether the performance values in this section, although we calculate the MAE for all models, are really comparable between the autoencoders and the other, supervised methods.

We further see that the two classification models RFclass and SVMclass perform better than their corresponding regression models (not only in median, but also regarding the minimum and maximum values). RNN is (in median) better than the simplest model LM, however, its interquartile distance is the largest among all models, closely followed by that of BA evaluated only on the target variable. Concerning NNA, there is not much difference whether the model is evaluated only on the target variable or on all reconstructed variables. This is unlike BA, where we can see a larger variance when we evaluate only on the target variable.

Although there are slight performance differences, the SVM and RF models all show only a small distance between their minimum and maximum performance values, i.e., these models seem to be quite stable and well generalizing independent of the respective Monte Carlo run. It is easy to see that the neural network based models show higher performance variances than the other models, where NNA and especially RNN exhibit large to very large outliers, i.e., really bad performing models. Regarding the exorbitantly high maximum value of RNN, we had a closer look on the respective Monte Carlo run. This high MAE is driven by one (an extremely bad one) of the 1950 predictions in the respective test set. Looking deeper at the corresponding wine features we see that this extremely bad prediction belongs to a wine with by far the highest value of “free sulfur dioxide” in the test set (two times as high as the wine with the second highest value) and the highest value of “total sulfur dioxide”. With an unlucky weighting of these features in the neural network, the extreme prediction value may be explained.

Regarding the times for hyperparameter optimization (not considering the sequential accumulative selection for RNN and NNA and not considering BA and LM here, as we did not optimize any hyperparameters for these two models) and training, we see the average of each model depicted in Table II. By far, the optimization and training time of RNN is the longest, where the fastest is that of RFclass. However, the grid for RNN is also larger than that of RF(class), thus, a mere comparison of the training times is not meaningful just like that. But keeping in mind that despite the relatively small grid, RF(class) performs better than RNN, RNN is no good choice for predicting the wine quality. It may be the case that other neural network architectures not captured by our hyperparameter grid would show a better performance than our RNN.

TABLE II
AVERAGE DURATION OF HYPERPARAMETER OPTIMIZATION AND TRAINING OF THE RESPECTIVE MODELS MEASURED IN MINUTES.

Model	Training Time (in minutes)
SVM	16.49
SVMclass	63.89
RF	16.79
RFclass	5.73
RNN	134.06
NNA	15.41

B. Manipulation Detection Performance

Next, we evaluate the detection performance of the non benchmark models. For this, we do set neither an explicit threshold for the share of data objects to be marked as suspicious nor an explicit threshold for the SE resp. SSE beyond which the data objects have to be marked as suspicious since the aim of this work is not to find a classifier for manipulated wine data quality but the comparison of the four resp. six models: SVM, SVMclass, RF, RFclass, RNN, NNA, where NNA's detection performance may be measured in two different ways (as before considering only the target feature or all reconstructed features). How a threshold can be found is, e.g., outlined in [53]. We first show and discuss the results of the first manipulation strategy with the corresponding test set $\underline{manip_{worst}}$ and then continue with the second manipulation strategy with the corresponding test set $\underline{manip_{random}}$. Note that in contrast to the general model performance analyzed in Section VI-A, the detection performance is unquestionably comparable between the autoencoder and the other models as we work with orderings here and not with absolute values, as it is the case when, e.g., comparing several MAE values. All statements in this section apply in tendency, as they depend on chance (especially the neural networks) and on q.

1) First Manipulation Strategy $manip_{worst}$: To illustrate the detection performance of the models mentioned above in the context of the manipulation of the 10% worst rated wines, we calculate tpr and fpr for all Monte Carlo like runs and for all $q_i = 1, \ldots, 99$. For all q_i, we calculate the five quartiles of tpr and fpr for each model and plot these values against q. The results are depicted in Figures 5 (tpr) and 6 (fpr).

As we can easily observe in Figure 5, all models are better than randomly guessing since all lines are above the diagonal. An optimal detection model would linearly increase and reach a tpr of 100% at $q = 10\%$, as already explained in Section IV-G. Among all models, the NNA (and NNA_all) is by far the worst model concerning the detection of the manipulated wines. This is interesting as NNA is, at the same time, the model performing best on the unmanipulated test data. Further, autoencoders are generally used for outlier detection, but here it seems that the regression models are more suitable for the detection of data manipulation in the target variable. The fpr in Figure 6 shows the corresponding behavior of the models. Here, an optimal model would have a fpr of 0 up to $q = 10\%$, i.e., there are no false positives among the first 10% of the data objects when sorted according to their deviance of predicted and actual (manipulated) values, and then linearly increase to the point $(100\%, 100\%)$. A summary of the medians of all models and all manipulation strategies at $q = 10\%$ is given in Table III.

2) Second Manipulation Strategy $manip_{random}$: We repeat the same analysis as in Section VI-B1 for the second manipulation strategy, where we randomly changed the target variable of 10% of the data to a plausible, but also random value. In Figures 7 and 8 we show the tpr resp. fpr on the manipulated test set $\underline{manip_{random}}$. Note that among the 10% data objects marked as manipulated, not all of them are necessarily manipulated. This is why we slightly adjust the analysis resp. the manipulation markings of the detection performance by marking only those data objects as manipulated where a manipulation has actually taken place ($\underline{manip_{random}} \mapsto \underline{manip_{random2}}$). This leads to a maximum manipulation rate of 10%, i.e., the actual manipulation rate is somewhere between 0% and 10% and depends on the respective Monte Carlo run.

The adjustment of the analysis is depicted in Figures 9 and 10 and works, as can be seen in the figures, in favor of the models, i.e., the adjustment increases their tpr and lowers their fpr. In the further course, we only discuss the results of the adjusted detection performance measurements.

When regarding Figure 9, it is immediately obvious that all models have more trouble detecting the manipulated data objects than it was the case for the first manipulation strategy. The order of the models concerning their detection ability is more or less the same as in Section VI-B1 but at a lower level. The best model shows a median tpr of about 37% for $q = 10\%$ where on $\underline{manip_{worst}}$ the median tpr at $q = 10\%$ is about 87% (see also Table III). Again, the NNA is the worst performing model, where its minimum tpr over all runs when measuring its detection performance only on the target variable is even worse than randomly guessing. Note that on $\underline{manip_{random2}}$, an optimal model would reach a tpr of 100% at the latest for $q = 10\%$ and possibly already for $q < 10\%$.

VII. CONCLUSION

We analyze the ability of different machine learning methods for detecting manipulated wine ratings. In detail, we

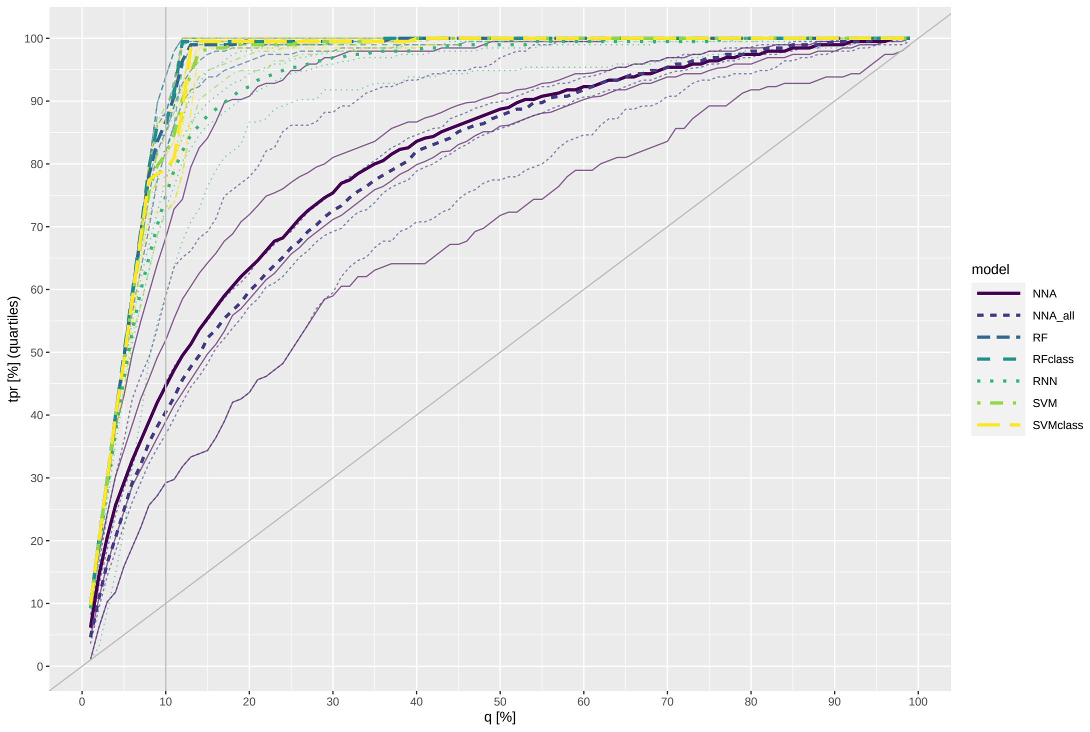

Figure 5. The five quartiles of tpr on $\underline{manip_{worst}}$ for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

TABLE III
MEDIAN DETECTION PERFORMANCE IN % OF ALL MODELS ON $\underline{manip_{worst}}$ (w) AND $\underline{manip_{random}}$ (r) RESP. $\underline{manip_{random2}}$ $(r2)$ FOR $q = 10\%$.

	tpr,w	tpr,r	tpr,$r2$	fpr,w	fpr,r	fpr,$r2$
RF	86.667	26.154	36.965	1.425	8.148	8.013
RFclass	86.154	24.615	33.835	1.481	8.319	8.256
SVM	81.538	23.077	32.046	1.994	8.490	8.329
SVMclass	78.974	21.538	29.720	2.279	8.661	8.551
RNN	74.872	21.538	29.458	2.735	8.661	8.505
NNA	44.615	15.385	18.919	6.097	9.345	9.304
NNA_all	40.513	14.359	16.288	6.553	9.459	9.491

consider autoencoders implemented via neural networks, neural network regressions, support vector machines used both for regression and classification, as well as random forests used both for regression and classification. We measure these models' general performance on an unmanipulated test set and their detection performance on two different manipulated test sets. The first manipulation strategy increases the quality ratings of those wines that were originally the worst rated. The second manipulation strategy changes the quality ratings of randomly picked wines to a value drawn from the empirical distribution of the remaining, unmodified part of the test data. As a benchmark for the models' general performance we additionally consider a linear regression and a minimalistic benchmark autoencoder. The latter two models are not used for the manipulation detection. All data splitting, training, and testing steps are repeated 100 times in a Monte Carlo like manner in order to get more robust results. Our case study is conducted on two vinho verde datasets.

We find that the more elaborate models generally perform better than their respective benchmark model. The autoencoders show the smallest mean absolute error (in median over all runs) on the unmanipulated test data, but it is not clear whether the performance measures are really comparable since the prerequisites for autoencoders and supervised models are not the same. Among the supervised models, the support vector model classification and the random forest classification (their results also measured via the mean absolute error) basically perform best. The neural network based models show a great variability across the different runs.

Concerning the models' ability to detect manipulated wine ratings, the supervised machine learning models outperform the autoencoder. For manipulation strategy one, the random forest models show a true positive rate of over 86% (in median) when marking the 10% test wines where the predicted quality and the (possibly manipulated) actual quality deviate most. As expected, all models show a better detection performance on manipulation strategy one (which we intended to be economically reasonable) than on manipulation strategy two (the random manipulation). Since the classification random forest is among the models with the best performance values and has the least optimization and training time (where the hyperparameter grid we use is quite sparse), it may be denoted as the most suitable model for the detection of manipulated

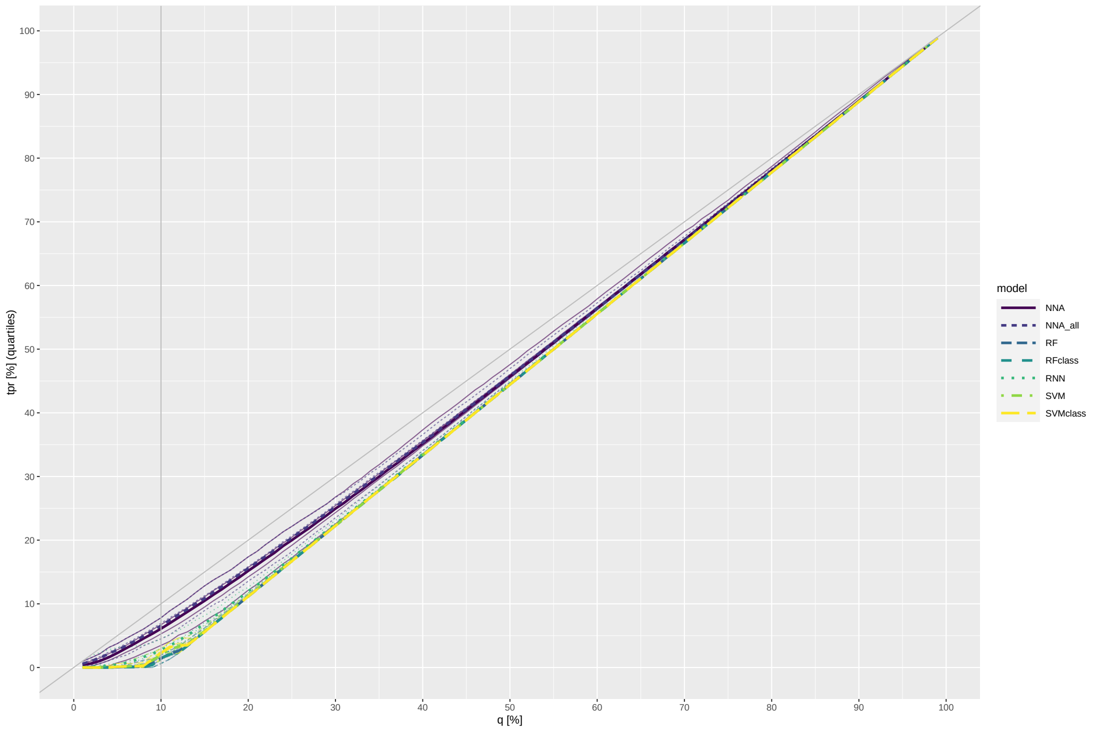

Figure 6. The five quartiles of fpr on $\underline{manip_{worst}}$ for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

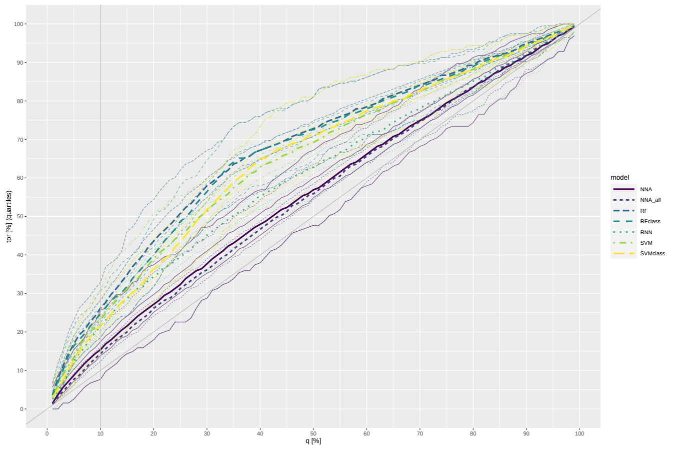

Figure 7. The five quartiles of tpr on $\underline{manip_{random}}$ for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

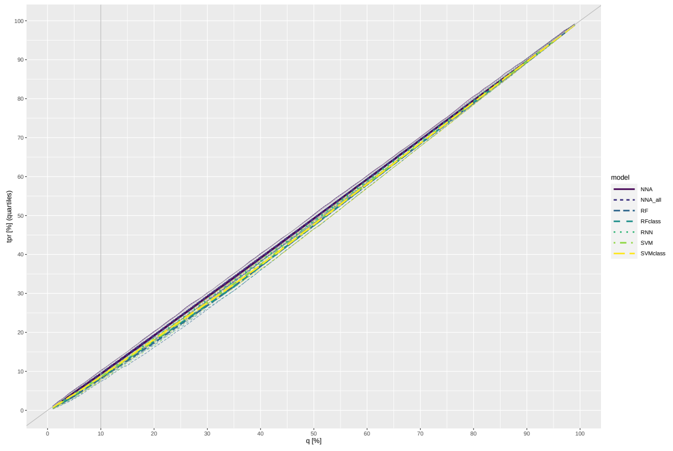

Figure 8. The five quartiles of fpr on $\underline{manip_{random}}$ for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

wine ratings.

In order to allow for a relatively large initial hyperparameter grid for the neural network based models (autoencoder network and regression neural network), we establish the procedure of sequential accumulative selection, where in a pre step the initial grid is sequentially reduced before the actual full grid search is applied. Despite the comparably large effort we put into the optimization of the neural networks, they show a great variability.

VIII. Future Work

In this paper, we first assume that it is reasonable that manipulations are applied to low rated wines to make them appear better to increase sales numbers. As a reference setting, we additionally apply a random manipulation. However, it would be interesting to test our approach also on other, somehow meaningful manipulation strategies, including, e.g., intentional and unjustified down ratings. Future work could also deal with the detection of faked ratings when there are multiple ratings

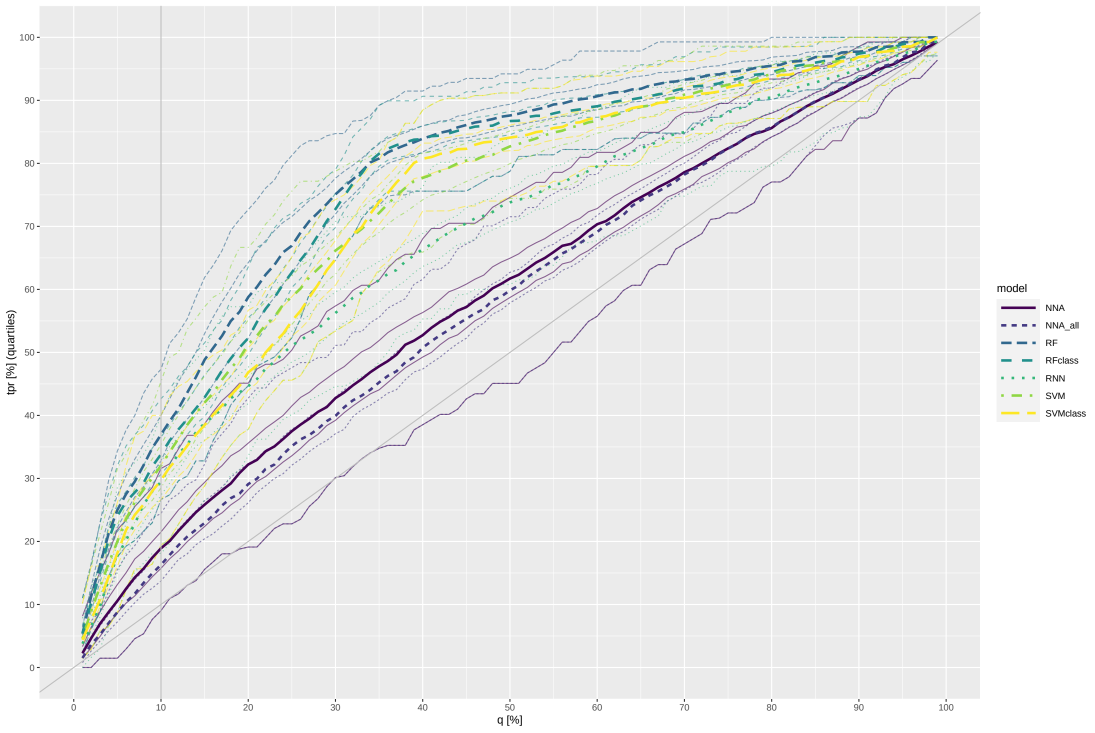

Figure 9. The five quartiles of tpr on $\underline{manip_{random2}}$ considering the actually manipulated data objects for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

per product as it is typical for many online stores or rating portals. Are there ways to detect the faked/manipulated ratings (whether better or worse) when there are many ratings for the same product? In this context, many stores and portals offer the possibility to write a review in addition to the plain rating. The processing of such information (via natural language processing) is likely to be useful here. In our approach, as it is typical for wine and other online ratings, we did not assume that cryptographic means for the prevention of manipulations exist. This is mainly because ratings can be manipulated before they are written down the first time. However, it could be interesting to check how the introduction of some cryptographic means, e.g., manipulation detection codes [55], which are some kind of checksums, could influence the detection of manipulated ratings.

Of course, other application areas apart from wine can be investigated with our approach, for example, ratings for products in online stores, restaurants, hotels. The detection of fraud in telecommunication, insurance, etc. [54] is also closely related. It could be of interest to identify the similarities and differences between these applications and how they should be addressed. When analyzing wine ratings, it would also be interesting to transfer our approach to other, larger datasets with more features, such as countries, producing regions, price segments, etc. and analyze the stability of the models' performances.

The procedure of *sequential accumulative selection* (as explained in Section V) can be further analyzed. One might investigate whether and how the order of the features is important. Comparisons to other hyperparameter selection methods are also possible (cf. [32]).

Last but not least, it should be noted that the topic of explainable AI and responsible AI is rapidly growing in importance [56]. It would be helpful to get to know the reasons why a manipulation detection model marks a certain data object as suspicious. As few as possible false positives are to be marked, whereas all manipulated ones are to be recognized if possible. So how can the decisions of the recognition algorithms be (understandably) explained?

Acknowledgment

Michaela Baumann is with NÜRNBERGER Versicherung, Germany. The opinions expressed here are her own and not necessarily those of her employer.

The authors thank Lars Grüne and Bernhard Herz, both with University of Bayreuth, Germany.

References

[1] M. Baumann and M. H. Baumann, “Autoencoder vs. Regression Neural Networks for Detecting Manipulated Wine Ratings,” The Seventeenth International Multi-Conference on Computing in the Global Information Technology (ICCGI), 2022, pp. 7-13
[2] G. Rosso, Italian Wines 2021 *(English Edition),* Gambero Rosso, 2021
[3] R. Parker, The Wine Advocate, https://www.robertparker.com/articles/the-wine-advocate, accessed: 2022.12.08
[4] Gault&Millau, https://www.gaultmillau.com/, accessed: 2022.12.08
[5] Guide Michelin, https://guide.michelin.com/en, accessed: 2022.12.08
[6] D. Freedman, R. Pisani, and R. Purves, “Statistics,” 4th ed., W. W. Norton & Company, Inc., New York, London, 2007, Chapters 10-12

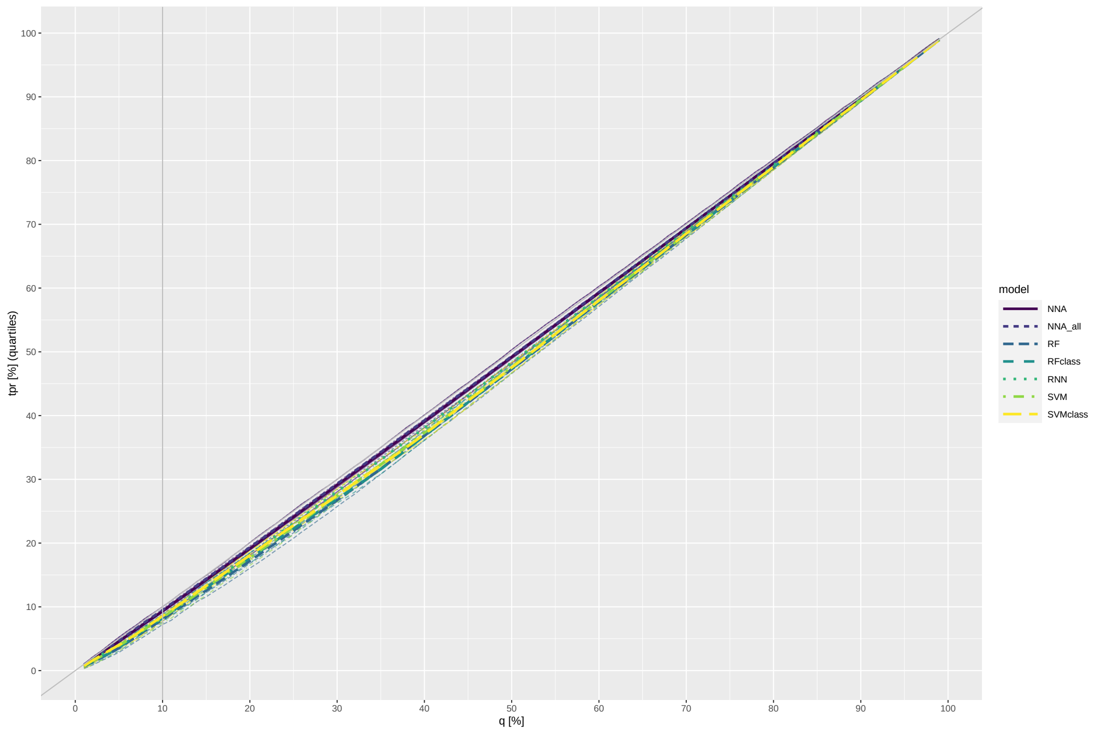

Figure 10. The five quartiles of fpr on $\underline{manip_{random2}}$ considering the actually manipulated data objects for varying q for all models distinguishable by color and line type. The median is drawn thicker. Additionally, the diagonal and the 10% line are depicted.

[7] E. Gelenbe, Z. H. Mao, and Y. D. Li, "Function Approximation with Spiked Random Networks," in IEEE Transactions on Neural Networks, vol. 10, no. 1, 1999, pp. 3-9

[8] E. Gelenbe, "Random Neural Networks with Negative and Positive Signals and Product Form Solution," in Neural Computataion, vol. 1, no. 4, 1989, pp. 502-510

[9] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and When Can Deep-but Not Shallow-networks Avoid the Curse of Dimensionality: A Review," in International Journal of Automation and Computing, vol. 14, no. 5, 2017, pp. 503-519

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," in Nature, vol. 521, no. 7553, 2015, pp. 436-444

[11] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support Vector Regression Machines," in Advances in Neural Information Processing Systems, vol. 9, 1996

[12] I. Steinwart and A. Christmann, "Support Vector Machines," Springer, 2008

[13] L. Breiman, "Bagging Predictors," in Machine Learning, vol. 24, 1996, pp. 123-140

[14] L. Breiman, "Random Forests," in Machine Learning, vol. 45, 2001, pp. 5-32

[15] S. Hawkins, H. He, G. Williams, and R. Baxter, "Outlier Detection Using Replicator Neural Networks," Data Warehousing and Knowledge Discovery, 2002, pp. 170-180

[16] M. Sakurada and T. Yairi, "Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction," Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis, 2014, pp. 4-11

[17] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," in Science, vol. 313, no. 5786, 2006, pp. 504-507

[18] J. D. Kelleher, "Deep Learning," MIT press, 2019

[19] M. A. Kramer, "Autoassociative Neural Networks," in Computers & Chemical Engineering, vol. 16, no. 4, 1992, pp. 313-328

[20] M. A. Kramer, "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks," in AIChE journal, vol. 37, no. 2, 1991, pp. 233-243

[21] D. L. Donoho, "High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality," in AMS math challenges lecture, 2000

[22] J. W. Tukey, "The Future of Data Analysis," in The Annals of Mathematical Statistics, vol. 33, no. 1, 1962, pp. 1-67

[23] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling Wine Preferences by Data Mining from Physicochemical Properties," in Decision Support Systems, vol. 47, no. 4, 2009, pp. 547-553

[24] P. Cortez et al., "Using Data Mining for Wine Quality Assessment," in International Conference on Discovery Science, Springer, Berlin, Heidelberg, 2009, pp. 66-79

[25] Y. Gupta, "Selection of Important Features and Predicting Wine Quality Using Machine Learning Techniques," in Procedia Computer Science, vol. 125, 2018, pp. 305-312

[26] À. Nebot, F. Mugica, and A. Escobet, "Modeling Wine Preferences from Physicochemical Properties Using Fuzzy Techniques," in SIMULTECH, 2015, pp. 501-507

[27] S. Kumar, Y. Kraeva, R. Kraleva, and M. Zymbler, "A Deep Neural Network Approach to Predict the Wine Taste Preferences," in Intelligent Computing in Engineering, Springer, Singapore, 2020, pp. 1165-1173

[28] P. Abbal, J. M. Sablayrolles, E. Matzner-Lober, and A. Carbonneau, "A Model for Predicting Wine Quality in a Rhône Valley Vineyard," in Agronomy Journal, vol. 111, no. 2, 2019, pp. 545-554

[29] O. Ashenfelter, "Predicting the Quality and Prices of Bordeaux Wine," in The Economic Journal, vol. 118, no. 529, 2008, pp. F174-F184

[30] O. Ashenfelter, "Predicting the Quality and Prices of Bordeaux Wine," in Journal of Wine Economics, vol. 5, no. 1, 2010, pp. 40-52

[31] R. Schwarz, "Predicting Wine Quality from Terrain Characteristics with Regression Trees," in Cybergeo: European Journal of Geography, 1997

[32] T. H. Y. Chiu, C. Wu, and C. H. Chen, "A Generalized Wine Quality Prediction Framework by Evolutionary Algorithms," in International Journal of Interactive Multimedia & Artificial Intelligence, vol. 6, no. 7, 2021, pp. 60-70

[33] R. Andonie, A. M. Johansen, A. L. Mumma, H. C. Pinkart, and S. Vajda, "Cost Efficient Prediction of Cabernet Sauvignon Wine Quality," IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1-8

[34] U. G. Mahima, Y. Patidar, A. Agarwal, and K. P. Singh, "Wine Quality Analysis Using Machine Learning Algorithms," Micro-Electronics and Telecommunication Engineering, Springer, 2020, pp. 11-18

[35] S. Bhattacharjee and M. R. Chaudhuri, “Understanding Quality of Wine Products Using Support Vector Machine in Data Mining,” in Prestige International Journal of Management & IT-Sanchayan, vol. 5, no. 1, 2016, pp. 67-80

[36] S. Kumar, K. Agrawal, and N. Mandan, “Red Wine Quality Prediction Using Machine Learning Techniques,” International Conference on Computer Communication and Informatics (ICCCI), 2020, pp. 1-6

[37] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly Detection: A Survey”, ACM Comput. Surv., vol. 41, no. 3, 2009, article no. 15, pp. 1-15

[38] V. Hodge and J. Austin, “A Survey of Outlier Detection Methodologies.” Artificial Intelligence Review, vol. 22, 2004, pp. 85-126

[39] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “Network Anomaly Detection: Methods, Systems and Tools,” in IEEE Communications Surveys & Tutorials, vol. 16, no. 1, 2014, pp. 303-336

[40] A. Patcha and J.-M. Park, “An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends,” in Computer Networks, vol. 51, no. 12, 2007, pp. 3448-3470

[41] R. Chalapathy and S. Chawla, “Deep Learning for Anomaly Detection: A Survey,” preprint on arXiv, https://arxiv.org/abs/1901.03407, 2019

[42] C. C. Noble and D. J. Cook, “Graph-Based Anomaly Detection,” Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 631-636

[43] Wine Quality Datasets, Universidade do Minho, http://www3.dsi.uminho.pt/pcortez/wine/, accessed: 2022.12.08

[44] kaggle (Piyush Agnihotri), White Wine Quality—Simple and Clean Practice Dataset for Regression or Classification Modelling, https://www.kaggle.com/piyushagni5/white-wine-quality, accessed: 2022.12.08

[45] kaggle (UCI Machine Learning), Red Wine Quality—Simple and Clean Practice Dataset for Regression or Classification Modelling, https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009, accessed: 2022.12.08

[46] Open Data Commons—Legal Tools for Open Data, Database Contents License (DbCL) v1.0, https://opendatacommons.org/licenses/dbcl/1-0/, accessed: 2022.12.08

[47] T. Shin, “Predicting Wine Quality with Several Classification Techniques” Towards Data Science, 2020, https://towardsdatascience.com/predicting-wine-quality-with-several-classification-techniques-179038ea6434, accessed: 2022.12.08

[48] D. Nguyen, “Red Wine Quality Prediction Using Regression Modeling and Machine Learning,” Towards Data Science, 2020, https://towardsdatascience.com/red-wine-quality-prediction-using-regression-modeling-and-machine-learning-7a3e2c3e1f46, accessed: 2022.12.08

[49] F. Rodríguez Mir, “Red Wine Quality,” Data UAB, 2019, https://datauab.github.io/red_wine_quality/, accessed: 2022.12.08

[50] *unknown,* “Wine Quality Prediction,” cppsecrets.com, 2021, https://cppsecrets.com/users/1012610010410511497106112114111106101996410310997105108469911111109/WINE-QUALITY-PREDICTION.php, accessed: 2022.12.08

[51] D. Alekseeva, “Red and White Wine Quality,” RPubs, https://rpubs.com/Daria/57835, accessed: 2022.12.08

[52] M. Hatz, “Der Einfluss von mtry auf Random Forests” (in English: “The Influence of mtry on Random Forests”), Master’s Thesis, 2018

[53] N. Japkowicz, C. Myers, and M. Gluck, “A Novelty Detection Approach to Classification,” in IJCAI, vol. 1, 1995, pp. 518-523

[54] M. Baumann, “Improving a Rule-based Fraud Detection System with Classification Based on Association Rule Mining,” INFORMATIK, 2021, pp. 1121-1134

[55] R. R. Jueneman, “A High Speed Manipulation Detection Code,” in Advances in Cryptology — CRYPTO’ 86, Lecture Notes in Computer Science, vol. 263, 1987, pp. 327-346

[56] M. Baumann, “Data Science Challenge 2021: Explainable Machine Learning,” https://github.com/DeutscheAktuarvereinigung/Data-Science-Challenge2021_Explainable-Machine-Learning, accessed: 2022.12.08