University of Warsaw
Faculty of Economic Sciences

Working Papers
No. 8/2020 (314)

NOVEL MULTILAYER STACKING FRAMEWORK WITH WEIGHTED ENSEMBLE APPROACH FOR MULTICLASS CREDIT SCORING PROBLEM APPLICATION

Marek Stelmach
Marcin Chlebus

Warsaw 2020

WORKING PAPERS 8/2020 (314)

Novel multilayer stacking framework with weighted ensemble approach for multiclass credit scoring problem application

Marek Stelmach*, Marcin Chlebus

Faculty of Economic Sciences, University of Warsaw
** Corresponding author: marekstelmach89@gmail.com*

Abstract: Stacked ensembles approaches have been recently gaining importance in complex predictive problems where extraordinary performance is desirable. In this paper we develop a multilayer stacking framework and apply it to a large dataset related to credit scoring with multiple, imbalanced classes. Diverse base estimators (among others, bagged and boosted tree algorithms, regularized logistic regression, neural networks, Naive Bayes classifier) are examined and we propose three meta learners to be finally combined into a novel, weighted ensemble. To prevent bias in meta features construction, we introduce a nested cross-validation schema into the architecture, while weighted log loss evaluation metric is used to overcome training bias towards the majority class. Additional emphasis is placed on a proper data preprocessing steps and Bayesian optimization for hyperparameter tuning to ensure that the solution do not overfits. Our study indicates better stacking results compared to all individual base classifiers, yet we stress the importance of an assessment whether the improvement compensates increased computational time and design complexity. Furthermore, conducted analysis shows extremely good performance among bagged and boosted trees, both in base and meta learning phase. We conclude with a thesis that a weighted meta ensemble with regularization properties reveals the least overfitting tendencies.

Keywords: stacked ensembles, nested cross-validation, Bayesian optimization, multiclass problem, imbalanced classes

JEL codes: G32, C38, C51, C52, C55

1. Introduction

In recent years, ensemble learning techniques have been constantly gaining in popularity for machine learning applications, especially in those areas where extraordinary predictive performance is recommended and problems addressed are of a rather complicated nature (among others, in Lessmann et al. (2015); Alaraj and Abbod (2016); Hung and Chen (2009); Marqués et al. (2012); Nanni and Lumini (2009); Xiao et al. (2016); Zhai and Chen (2018); Whalen and Pandey (2013); Madasamy and Ramaswami (2018); Zhang et al. (2018); Lessmann et al. (2019); Kang et al. (2015); Menahem et al. (2009)). The general idea is to use multiple algorithms and combine their predictions into a single one. We may divide them into homogenous ensemble models: in those methods weak learners from the same family are used (e.g., decision tree in Random Forest and Gradient Boosting Machine, see Hastie et al. (2009); Wan and Yang (2013); Brown and Mues (2012)) or heterogeneous ensemble models which take advantage of diverse set of learners and multilayer architectures, with stacking and blending being the most successful (e.g., in Shahhosseini et al. (2019); Zhang et al. (2018); LeDell (2015); van Veen et al. (2015)). As pointed out by LeDell (2015) the superior results do come with a burden of computational cost assigned to training multiple models and becomes more acute in case of rigorous validation and complicated frameworks present, therefore one should remember about the possible tradeoff between the score obtained and time consumption.

Among numerous ensemble approaches, stacking, in one of its forms sometimes also referred to as Super Learning (van der Laan et al. (2007)), appears to be the most widely used and achieves top performance improvements, notably those reported in machine learning competitions and literature (e.g., in Zhai and Chen (2018); Whalen and Pandey (2013); Sill et al. (2009)). With many available and evolving variations designed to solve particular problems from various fields, theoretical basis underlying the stacking structure consist of adapting cross-validation for training and selecting multiple algorithms - base learners - and building the final model - meta learner - on top of them, thus reducing the generalization error (Wolpert (1992)). In case of nonlinear, very complex relationships existing in the given data, when single classifiers may struggle to approximate the true prediction function (LeDell (2015)), stacked ensembles are often utilized. The advantages are even more tangible for broader area of data challenges, including multiclass or multilabel classification, large datasets or class imbalance which are currently regularly faced by many data science and business experts (among others, in Büyükçakir et al. (2018); Zhang et al. (2018); Madasamy and Ramaswami (2018); Sjardin et al. (2016); Kang et al. (2015); Menahem et al. (2009)).

An important area of studies that incorporates both high class imbalance and large data volumes, and is increasingly addressed by modern machine learning algorithms (Alaraj and Abbod (2016); Lessmann et al. (2015)), relates to broadly defined credit scoring, a topic essential from the perspective of financial institutions and proper loan decisions (Vahid and Ahmadi (2016)). Though numerous researches on credit scoring with ensemble methods utilized are available, they mainly focus on a traditional default prediction subject with good and bad entities distinguished (e.g., in Xiao et al. (2016); Brown and Mues (2012); Nanni and Lumini (2009)). Nevertheless, as shown by Irawan and Samopa (2019), Vahid and Ahmadi (2016) or Kwon et al. (2013), ensemble learning and stacking in particular might be successfully adapted to more complicated, multiclass creditworthiness assessment problems, for example existing in a form of credit ratings or various credit states.

The aim of this study is to propose a comprehensive stacking solution and investigate its performance against multiclass classification problem for peer-to-peer loan default dataset with over one million observations and highly imbalanced classes present. To obtain satisfactory results, three stage approach is proposed with nested cross-validation and Bayesian optimization for hyperparameter tuning and meta features construction. For the first stage multiple and diverse algorithms are used with decision tree based ensemble methods (e.g., bagging, boosting), linear classifiers (e.g., logistic regression, support vector machines), feedforward neural networks and Naive Bayes classifiers being the main ones. For the second level, three different meta learners (logistic regression, tree based model and neural network) are applied and it is further examined whether the final stage prediction being an ensemble of those three with corresponding weights tuned improves the whole solution effectiveness. Results are also compared with single base learners scores and we additionally observe if a model family outperforming others can be distinguished. In particular, the problems of overfitting, choosing the right evaluation metric and data preprocessing steps (e.g., dealing with high cardinality features) are addressed as well.

This paper is organized as follows. The second section contains literature review regarding different stacking approaches over the years and related work. Third section provides thorough methodology description, including algorithm architecture, data preprocessing and evaluation. Subsequent section describes the dataset used in research. In fifth part the empirical results are presented and the last section concludes the paper findings.

2. Literature review

The concept of stacking has been initially proposed by Wolpert (1992) with respect to neural networks as base learners. Stacked generalization idea, as introduced by the author, relies on adapting leave-one-out cross-validation to construct the - so called - level one data and combining given set of predictors instead of just selecting the best one. It is also concluded that picking the type and the number of generalizers (therefore defining the stacking architecture) is rather an art not a theoretically proven approach, which is, to some extent, the case even in modern machine learning solutions. Breiman (1996), by suggesting decision trees (with varying number of terminal nodes) and generalized linear models as base models, and k-fold cross validation framework (as opposed to Wolpert, mainly due to computational efficiency), extends Wolpert's research to develop stacked regressions. Although his stacking attempts are still made within the same model classes (CART or GLM) author achieves the most satisfying results from the mix of subset regressions (different predictor variables) and ridge regressions with different ridge parameters thus finally supporting a thesis that the most noticeable error reduction occurs when stacked models are not too similar (see also Hashem (1996); Lanes et al. (2017)).

Van der Laan et al. (2007) were among the first to provide a theoretical background for stacked ensembles and proposed a super learner algorithm that according to the authors research is an asymptotically optimal learning system (it performs asymptotically as well as the best estimator from given candidates). Super learner, composed of wide variety of candidate learners (e.g., CART, Random Forest, LARS or ridge regression), uses 10-fold cross-validation to obtain candidates predictions and on top of that fits a linear regression model with least squares method for learners parameters estimates (they may be interpreted as indicators of a given base model importance for the particular problem). In the study algorithm is applied to many different datasets performing reasonably well for each of them and therefore providing a strong evidence of its flexibility and stacking potential in general. Those observations for regression problems are confirmed by more modern researches, for example presented by Zhai and Chen (2018) who introduce a complex stacked generalization strategy to predict the PM2.5 concentration averages in China. Their approach, including, among others, XGBoost or AdaBoost (base layer) and Support Vector Regressor (meta learner), outperforms any single estimator considered. What is more, lower overfitting vulnerability is observed which is also the point in van Veen et al. (2015).

Another important study was conducted by Whalen and Pandey (2013) in the area of computational genomics where, as stressed by authors, even small performance improvements compared to the industry baseline are of a great importance, especially when one takes into account extreme class imbalance, high missing values percentage and feature complexity presence. Four different ensemble techniques are compared in the experiment: simple averaging of classifier's predictions, novel cluster based meta learning, ensemble selection based on choosing the best classifiers subset and stacking with logistic regression, with the latter offering the most significant improvement in the final results. Each technique is trained using up to 27 heterogeneous models with nested cross-validation and undersampling underneath. Advantages of adapting stacked ensembles to handle highly imbalanced datasets for classification problems are also outlined by Madasamy and Ramaswami (2018) and Zhang et al. (2018). First paper applies the two layer solution for the real-time data while the latter tests the similar architecture for plenty of different datasets and many metrics observed, both reporting impressing effectiveness. LeDell (2015), who introduces an AUC-maximizing super learner, claims that for a binary classification the gain obtained when compared to the top base algorithm is even higher if the class imbalance is more evident. Although the results itself are reliable in case of meta learning superiority, it is widely discussed whether AUC can be considered a good metric for skewed datasets (e.g., in Jeni et al. (2013)).

Beside choosing the proper evaluation metric for a particular case one should take into account rigorousness of cross-validation schema (e.g., whether it is flat, nested, repeated or combination), hyperparameter tuning and potential overfitting presence – crucial components with respect to designing the proper stacking architecture, though not directly referred to stacking *per se*. Cawley and Talbot (2010) along with Krstajic et al. (2014) emphasize the importance of rigorous cross-validation (both for model selection overfitting prevention and reliable performance evaluation), particularly with limited size datasets. On the other hand, Wainer and Cawley (2018) show that in practice using flat or standard cross-validation (even though it introduces bias) versus nested variant leads to the selection of models that are approximately almost the same in quality, provided there are relatively few hyperparameters to be tuned. To overcome the issues observed by aforementioned authors, Shahhosseini et al. (2019) propose a novel ensemble technique (COWE-ITH) based on nested algorithm and Bayesian optimization for hyperparameter tuning which is to a certain degree similar to our approach.

3. Methodology

3.1. Stacked ensembles

For this study we focus on meta learning with stacking, i.e., probabilistic outputs from base classifiers on the first level are further used as an input (meta features) to the meta classifier (Whalen and Pandey (2013)). In this form it is rather a standard, well known technique. The general solution is therefore extended by implementing nested cross-validation at the meta features creation level, introducing Bayesian optimization to effectively tune hyperparameters and redefining the meta classifier composition to finally come with a more innovative stacking design. The detailed architecture is presented in subsequent sections.

3.1.1. Cross-validation framework for stacked ensembles

The general aim of nested cross-validation is to obtain more reliable, almost unbiased estimator performance assessment, which is, according to Krstajic et al. (2014), a completely different task than using conventional cross-validation for model selection process. Having that knowledge, we use the mentioned property of nested cross-validation in order to build a meta features set that will not be biased towards any of the particular base learners. In other words, classifier's probabilistic predictions at the first, base stage of learning are considered to reflect the true estimation error. Pseudocode for the stacking algorithm is shown in Figure 1.

Algorithm 1: Stacking algorithm

1. split main dataset into $train_{80}$ and $test_{20}$ with 80:20 split ratio
2. for b=1 to B (number of base classifiers):
    2.1. split $train_{80}$ dataset into k_{outer}=3 stratified folds
    2.2. for each $training_{outer}$ fold (outer loop):
        2.2.1. split $training_{outer}$ into k_{inner}=3 stratified folds
        2.2.2. for each $training_{inner}$ fold (inner loop):
            2.2.2.1. $tune\_hyperparameters_{inner\_b}$
        2.2.3. fit $base\_classifier_b$ with chosen $hyperparameters_{inner\_b}$ to $training_{outer}$
        2.2.4. predict on $validation_{outer}$ and set aside $predictions_{outer\_b}$ for each class
        2.2.5. $tune\_hyperparameters_{outer\_b}$
    2.3. fit $base\_classifier_b$ with chosen $hyperparameters_{outer\_b}$ to $training_{80}$
    2.4. predict on $test_{20}$ to obtain the final score for $base\_classifier_b$
    2.5. merge 2.2.4. results across rows
3. merge 2.4. and 2.5. results across columns to obtain $train_{meta}$ and $test_{meta}$, respectively
4. for m=1 to M (number of meta classifiers):
    4.1. split $train_{meta}$ dataset into k_{outer}=3 stratified folds
    4.2. for each $training_{outer}$ fold:
        4.2.1. $tune\_hyperparameters_{outer\_m}$
    4.3. fit $meta\_classifier_m$ with chosen $hyperparameters_{outer\_m}$ to $train_{meta}$
    4.4. predict on $test_{meta}$ to obtain the final score for $meta\_classifier_m$

Figure 1: Pseudocode for the stacking algorithm.

Whole architecture consists of two loops: outer and inner, with training and validation folds created for both. First loop is used for model selection while the latter is responsible for hyperparameter tuning for meta features creation purposes. The choice of 3 x 3 nested cross-validation schema (k_{outer}=3 outer folds and k_{inner}=3 inner folds for each outer fold) is primarily dictated by the computational time reduction willingness and sufficient dataset size in which case there is no need for more rigorous approach as model selection variance is reduced (Cawley and Talbot (2010)). The $tune\_hyperparameters_{index}$ term, introduced for pseudocode readability, simply stands for hyperparameter tuning process consisting of averaging scores across all validation folds in cross-validation loop (whether it is inner or outer, for given base

model b or meta model m, all of them denoted by *index* subscript) and selecting hyperparameters set maximizing the average. Aforementioned best set obtained during the inner loop is then used to fit the model to $\text{training}_{\text{outer}}$ fold (which is actually repeated k_{outer} times) and predict on $\text{validation}_{\text{outer}}$ dataset (in literature also called out-of-sample or holdout data). This is in contrast to tuning and predicting only on $\text{training}_{\text{outer}}$ and $\text{validation}_{\text{outer}}$ (without nested folds) where the same data is used to search for the best model and estimate its performance. The detailed framework for hyperparameter tuning is presented in Figure 2.

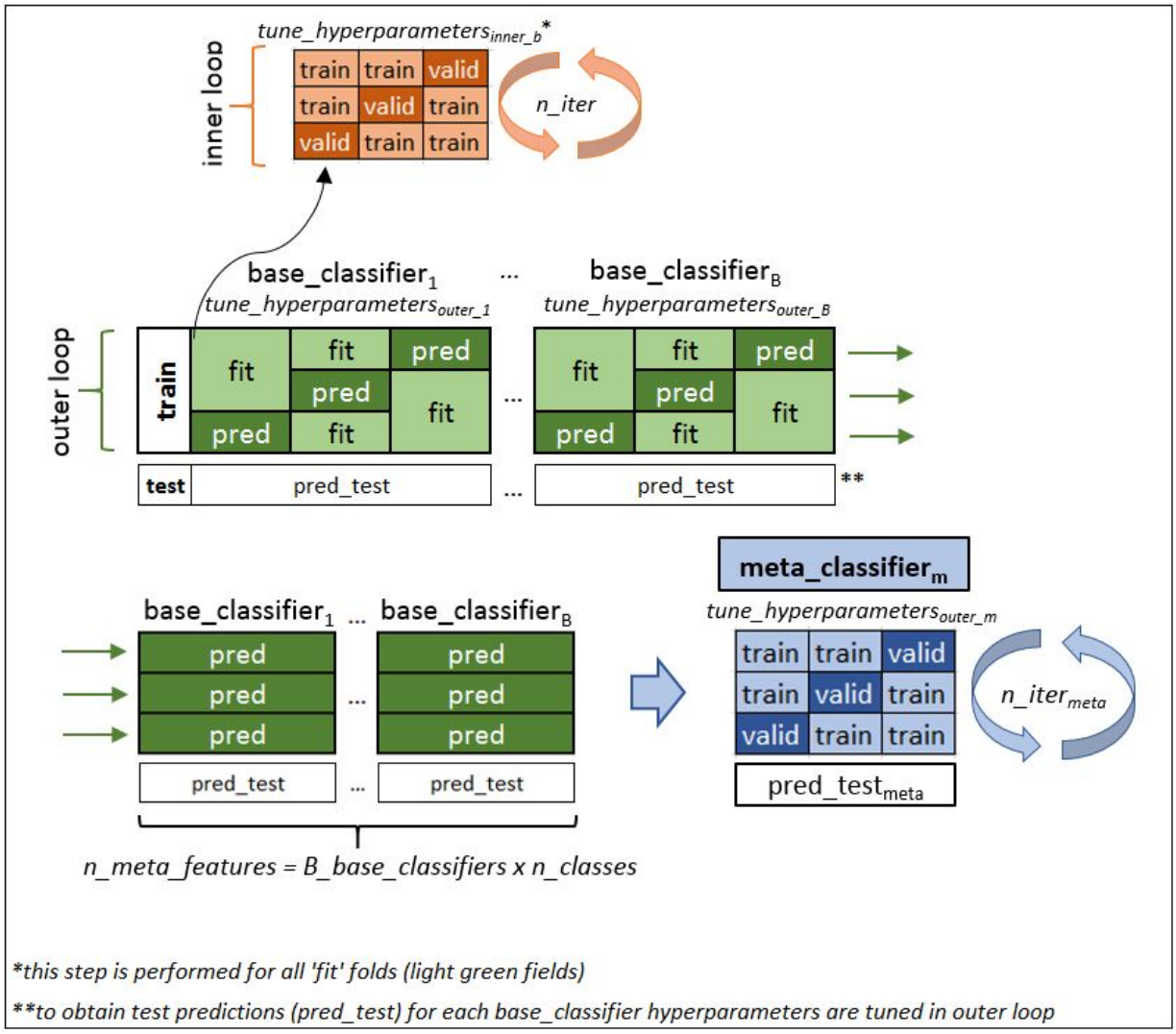

Figure 2: Hyperparameter tuning within outer and inner cross-validation loop.

Note: For each base_classifier_b from the pool of B base classifiers, hyperparameters are first tuned within the inner cross validation loop ($tune\_hyperparameters_{inner\_b}$) for *n_iter* iterations and the prediction 'pred' is made as a part of meta features set. Second, there is a hyperparameter tuning for the outer loop performed with the same number of iterations ($tune\_hyperparameters_{outer\_b}$) to obtain test set predictions ('pred_test'). Finally, meta features (blue fields) are used for meta_classifier_m hyperparameter tuning for n_iter_{meta} rounds ($tune\_hyperparameters_{outer\_m}$) resulting in predictions for $\text{pred\_test}_{\text{meta}}$ set, which is a concatenation of 'pred_test' sets for all base classifiers.

One may notice that there are actually two tuning rounds for outer folds performed. To be technically precise, first one is also extended by inner tuning stage for all base models. The key note at this point is to use the same folds across all trained classifiers. For simplicity let's assume that we train identical estimators on different folds and generate predictions for meta learning. It is rather obvious that a meta learner will choose the base estimator with a higher score, though they are both the same. It complicates even more if various models are fitted to different folds, thus introducing potential bias to the system. Nevertheless, diverse splits may be possible and beneficial but in case the repeated cross-validation is applied (Džeroski and Ženko (2004)). The second run is specifically designed for meta learners training with $train_{meta}$ dataset of *nrow*($train_{80}$) x (B**n_classes*) dimension, where B is the total number of base classifiers used and *n_classes* denotes dependent variable unique levels count.

Another important issue that arises is rather a practical concern and might have the effect on the total time spent on system training. Hyperparameter tuning, especially in case of complex algorithms, is rarely a one-time process – quite often a few rounds of tuning (e.g., with different parameter ranges) are required for results to be satisfactory. For that reason, any effective and flexible stacking architecture should be designed in a way allowing to work on a specific algorithm without the necessity to rerun already tuned estimators or the whole system. Thus, the pseudocode presented in Figure 1 begins with the loop iterating over base classifiers as opposed to initial cross-validation folds split followed by tuning all classifiers in one go.

The last point, also emphasized by Cawley and Talbot (2010), refers not only to particular stacking algorithm but to machine learning in general. In order to overcome the selection bias or any potential pitfalls during the algorithm building process, one should not treat it simply as fitting a specific model (like decision tree) to a new data but as a whole learning procedure which involves model selection and fitting integrally conducted. Considering the complexity of majority stacking approaches the latter is especially important for creating a successful solution.

3.1.2. Learners selection overview

Diverse predictions of base learners unleash the meta classifier's capability to collectively capture those areas where their performance on the holdout set is ultimate, leading to a more generalizable final ensemble and expected improved performance over each single first level estimator (Zhai and Chen (2018)). Therefore a proper attention is necessary in the algorithms

selection process and things like a specific dataset, computational time and model's characteristics need to be examined.

Considering all of the above we first construct the baseline approaches being the easiest to implement algorithms available. This consists of fitting a *dummy* classifier (it uses class frequencies as predictions) to have a general benchmark of other method's performance followed by training a relatively fast and simple Complement Naive Bayes model (Rennie et al. (2003)), an extended version well suited for handling imbalanced classes problems. The second one serves not only as a good point of reference but may also have an ability to build up valuable meta features.

For each subsequent estimator chosen (including naive bayes mentioned) mean cross-validation score obtained for the outer loop is reported and saved for further comparison purposes. What is more, the performance across all folds and tuning iterations is being tracked to observe the classifier's behavior on training and validation samples. Although time consuming in case of complex algorithms, such approach allows to tune hyperparameters more precisely, for instance by consecutively narrowing hyperparameter ranges or observing regions where model overfits and thus providing more reliable out-of-sample predictions. Additionally, the $train_{80}$ and $test_{20}$ datasets are scored, though only for the final performance reporting, not any kind of selection.

Based on the knowledge above, we are able to effectively discard algorithms performing worse than the baseline set, prone to overfitting or those with computational time inadequate to gains, in particular ones endlessly computing due to large dataset. It is also worth noting that any sort of comparisons should be especially rigorous among classifier's families and not as such between families themselves. As an example, one may almost harmlessly reject one of two variations of logistic regression with similar score, while throwing away tree based algorithm only because it performs worse than logistic regression might be unreasonable due to the fact that, despite the weaker score, different patterns are potentially discovered.

To check for learners diversity we also examine the correlation matrix for meta features dataset but it is conducted rather to make sure about its presence (at least to the extent the correlation defines diversity) than to discard base models only due to high correlation for a given class predictions. As Lanes et al. (2017) indicate, even though the meta set is considered more diversified (whichever metric is chosen), the stacking solution itself does not necessarily improve. We should then consider diversity as a set of different patterns captured by mixed classifiers, not a single measure. To ensure the aforementioned is provided, ensemble based

methods, linear classifiers, neural networks, support vector machines, naive bayes and k-nearest neighbors are investigated and trained, with the main focus on the differences between groups of algorithms and algorithms themselves as well.

From among boosting methods, one of ensemble families, we choose to adapt Gradient Boosting Machine (GBM) introduced by Friedman (2000), multiclass extension of AdaBoost called SAMME (Zhu et al. (2009)) and a very competitive XGBoost (Chen and Guestrin (2016)). Those three, despite being designed for sequential optimization with weak learners, differ in approaches. GBM utilizes regression trees fitted on top of the gradient of the loss function deviance. On the other hand, SAMME uses classification trees (at least in our research) in a stagewise additive modeling manner minimizing the exponential loss. XGBoost, the most recent among considered propositions, has a more regularized formalization than traditional gradient boosting and scales incredibly well (sparsity awareness, multicore computations, in Chen and Guestrin (2016)).

Another representation of ensemble methods, also referred to as averaging, unlike boosting builds a set of independent estimators aggregated in order to reduce variance (Hastie et al. (2009)). In this paper two decision tree based algorithms are presented: Random Forest (Breiman (2001)) and Extra-Trees (extremely randomized trees, in Geurts et al. (2006)). Each of them places the emphasis on rather deeply grown trees and randomization (bootstrap samples, random subsets of features) as a means to variance reduction, but Extra-Trees moves one step further in its randomness and chooses the best split in particular tree node amongst randomly designated splits.

Turning to linear classifiers, first of all we focus on training multinomial logistic regression extended by the elastic net regularization included to prevent overfitting (Friedman et al. (2010)). As an alternative, stochastic gradient descent algorithm (SGD) with modified Huber loss is proposed (Sjardin et al. (2016)). Although both similar in the core concept, they vary in optimization method and loss function underneath thus leading to potentially different predictions. Following the linear classification, we also apply Support Vector Machine (SVM) model with linear kernel and multiclass probabilistic estimates calculated according to Wu et al. (2004).

Several architectures of feedforward neural networks using back-propagation are fitted as well, with combinations of hidden layers number and size, activation functions (tanh, relu, sigmoid), dropout, L1 and L2 regularization, diverse optimizers (e.g., SGD, Adam) and learning schema (Geron (2017)). Nevertheless, we keep in mind that complex structures are more

computationally expensive and the gain achieved should be satisfactory in comparison with simpler algorithms.

Already mentioned Support Vector Machine is also trained with polynomial and Gaussian RBF kernels to capture nonlinear relationships, however, as stated by Geron (2017), neither of presented versions is predestined for large datasets. The same applies to K-Nearest Neighbors classifier (KNN; tried with various neighbors numbers), even though faster implementations are used in our research (Hastie et al. (2009)).

For meta learners, three algorithms from different families are chosen arbitrarily: SGD optimized logistic regression with regularization (Sjardin et al. (2016)), feedforward neural network and tree based method (XGBoost or Extra-Trees – to be decided during ensemble performance investigation). Those are tuned and assessed separately within the second phase of stacking as discussed in Section 3.1.1. The novel concept that we introduce is to combine all meta models into a single big estimator (according to Figure 1 pseudocode it might be considered as M-th meta classifier) and tune all hyperparameters expanded by corresponding meta weights w_1, w_2 and w_3 simultaneously (approximately 20 hyperparameters in total). This is what we call the third stage of training, although technically it is proceeded in parallel to the second one. Each tuning iteration then consists of fitting a set of hyperparameters (including w_1, w_2, w_3) with the final prediction for a given class being a weighted average of constituent models predictions, thereby reducing the stacking variance. We believe that in case there is no a single meta learner that dominates others in terms of estimation accuracy, hyperparameters chosen and performance is expected to be different than if tuned independently (as in the second stage), which would be in accordance with Shahhosseini et al. (2019).

To conclude, dependent on the particular algorithms specifications, we cross-validate roughly up to 20 base classifiers, which is a sufficient and reasonable amount according to the literature provided. Nonetheless, due to selection process described, not all of them may be chosen for the second and third phase of stacking where four learners are fitted altogether with the last constructed as an aggregated hybrid ensemble.

3.1.3. Hyperparameter tuning with Bayesian optimization

Taking into account the complexity level and the total number of algorithms applied in this research, one needs to focus on time efficient and reliable method leading up to finding the best hyperparameters sets. This term refers not only to algorithm specific parameters, but also to

weights used for training to overcome the class imbalance problem and weights associated with meta learners aggregation (see Section 3.1.2. for more details), as those are all tunable as well. For the above reasons, we choose to adapt the Bayesian search for hyperparameter optimization, since its effectiveness is widely reported in the literature (among others, in Shahhosseini et al. (2019); Snoek et al. (2012); Rasmussen and Williams (2006)), especially in the event of large parameters space scanned.

In referring to Snoek et al. (2012), the general idea is to approximate the underlying, unknown function, usually by utilizing Gaussian processes, based on initial random guesses and to update it at each step when the most recent information (i.e., new observation) is being gathered. In our case, a function denotes evaluation measure chosen. The simplified form of such process is presented in Figure 3.

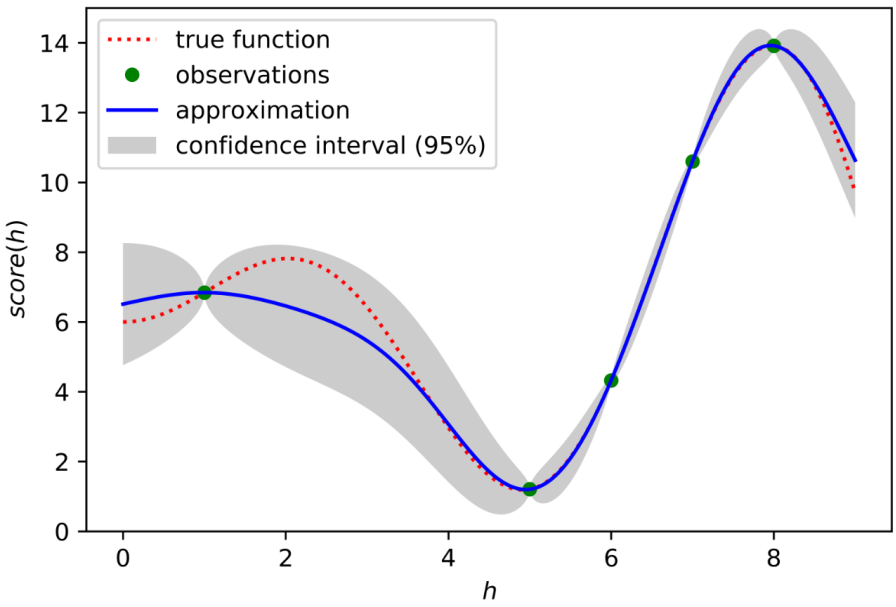

Figure 3: Bayesian optimization of score function for a single hyperparameter value h and 5 iterations.

Note: True *score* (assumption is that it exists and can be defined in a functional form) is well approximated in the area that has been *exploited* by more observations (h=5 to 8), while the region around h=1 is still to be *explored*. Values on the axes serve only as an example.

Up against multi-dimensional space with many hyperparameters available, sufficient number of iterations and careful hyperparameter candidate ranges selection is crucial for stacking success, though the exact numbers are rather dictated by the dataset size, expert judgement, intuition and computational resources. Our algorithms, particularly the most

sophisticated ones like the weighted meta learner trained during the third stage, are tuned with even up to 100 iterations, not to mention multiple tuning rounds performed for vast majority of them. This consumes meaningful amount of time, but is still considered to be more efficient in terms of obtaining the optimum within fewer trials than traditional random search (e.g., in Nishio et al. (2018); Yamashita et al. (2018)).

3.2. Data preprocessing and feature engineering

This section summarizes all data preprocessing and feature engineering or extraction techniques applied in our research, with some of them constituting a standard in machine learning implementations whilst others being rather innovative and hybrid approaches. Furthermore, several steps are only applicable for certain types of algorithms, which is also outlined. To prevent overfitting and ensure a good ensemble generalization, we avoid any potential data leakage, meaning each rule that requires learning is trained without the contribution of validation or test datasets, in particular during cross-validation loops.

After initial data cleaning that includes, among others, missing values patterns investigation according to Zhang (2015) and dropping (almost) perfectly correlated or (near) zero variance variables, we use the following methods, presented in order of their actual application.

Power transform

In case of highly skewed distributions, Yeo-Johnson transformation is employed as it allows, through maximum likelihood estimation, to make the data more normal-like and is also dedicated for nonpositive values (Yeo and Johnson (2000)). Although tree based models, which constitute a significant part of our project, are rather insensitive to predictors skewness, Gaussianity is recommended for many other algorithms (Kuhn and Johnson (2013)). Additionally, a symmetric distribution combined with data scaling is also a useful property for subsequent distance based engineering methods.

Word2vec and k-means clustering

Raw text inputs, for instance customers answers gathered during credit interviews, are handled by word2vec and k-means algorithms altogether – the approach we call *sentences to*

variable, conceptually similar to a sentiment expression analysis performed by Wang and Castanon (2015). In general, upon initial text cleaning, a word2vec learns the numerical representation of the entry tokenized strings (Mikolov et al. (2013)) and k-means clusters the received sentences vector space (word vectors are averaged across sentences at first) to eventually extract a single, cluster labeled categorical variable. Specifically, we expect to obtain a separate cluster for empty input sentences with others carrying a meaningful and differentiated, word based information. Technically, for word2vec a fifty-dimensional vector size is defined, whereas k-means relies on Euclidean distance and an automatic algorithm specifying the optimal number of clusters (H2O.ai., n.d.).

Target encoding

To deal with high cardinality categorical features like zip codes or addresses, we incorporate a mean aggregation of target variable in relation to the problematic feature, as proposed by Micci-Barreca (2001). This approach is then extended for multiclass classification and the calculation is performed in one versus all manner with a given level of dependent variable denoted as 1 and 0 assigned to the other instances (analogical example for WoE in Zdravevski et al. (2015)). Therefore, we end up with *n_classes* new attributes. The target encoding formula calculated for class *c* and categorical predictor's unique level *ul* is presented in Equation 1.

$$S_{ul} = \lambda(n_{ul})\frac{n_{c\_ul}}{n_{ul}} + \big(1 - \lambda(n_{ul})\big)\frac{n_c}{n_{tr}}, \tag{1}$$

where:

S_{ul} = calculated estimate within a [0, 1] range,

λ = sigmoidal function (smoothing parameter),

n_{ul} = number of records for unique level *ul*,

n_{c_ul} = number of class *c* positive instances for level *ul*,

n_c = total number of class *c* positive instances,

n_{tr} = total number of training set records.

Apart from introducing a blended average (λ parameter) to prevent overfitting, the desired statistic is also calculated in a 3-fold cross-validation scheme for each training set, with the final estimate being an average aggregated only from holdout samples. Missing values are left alone for further processing, they are not encoded.

Missing data imputation

Although some of modern machine learning algorithms have their own, in-built imputation methods (e.g., XGBoost), we decide to choose a unified line throughout all models and apply a Random Forest multiple imputation technique, mainly due to its effectiveness reported (e.g., in Shah et al. (2014); Tarap and Stos (2019)). As shown by Shah et al. (2014), continuous features, becoming respectively response variables for the sake of algorithm training process, are imputed with random values drawn from Gaussian distribution that is centered on the Random Forest mean prediction and scaled by the out-of-bag root mean square error (RMSE) obtained from a given fit. For categorical attributes, authors propose to train numerous independent tree estimators on bootstrap samples and predict each missing value with a randomly selected tree. Before this step, we additionally encode a separate *NA* category for the highest missing percentages observed.

KNN and K-means based features

Classifiers performance is potentially enhanced by the implementation of K-Nearest Neighbors as a feature engineering method. For the specified number of neighbors *k* and within each dependent variable class separately, we compute a sum of distances between a given observation and its 1 up to *k* nearest neighbors, thus resulting in *k* n_classes* new features that characterize in transition from a nonlinear space into a linear one (Pinto (2017)). For our research *k*=5 is chosen to control the total preprocessing time.

K-means, on the other hand, allows us to engineer two types of predictors – nominal one, represented by the clusters labels, and *K* numerical features calculated as distances from each cluster center, where *K* is the total number of clusters. The same technical specification as previously described in word2vec section is used.

Prior to the application of either aforementioned technique, the input data is subset only to continuous, power transformed variables and then standardized.

In addition to outlined preprocessing steps, before feeding the data to a base learner and if it is advised for the given estimator to do so, continuous features are scaled, either by normalization ([0, 1] range) or standardization (zero mean, unit variance). In particular, this

refers to a family of stochastic gradient optimized algorithms. Furthermore, remaining categorical variables, if not already target encoded, are treated by a one-hot encoder.

3.3. Model evaluation

To properly address the class imbalance problem and build a reliable meta features set (constructed from probability outputs, see Section 3.1.1.), special emphasis should be placed on a performance measure that handles these both requirements. To fulfil the second condition, we choose a log loss metric, as it allows to evaluate the quality of the probabilistic estimates and prepare a good input for an ensemble algorithm (Ferri et al. (2009)). The target classes skewness is addressed by introducing case weights, which is shown in Equation 2 with a modified log loss formula for multiclass classification (often referred to as cross entropy; in Geron (2017)) implemented in our solution.

$$LogLoss = -\frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} \sum_{c=1}^{C} w_i * y_{i\_c} * ln(p_{i\_c}) \tag{2}$$

where:

n = number of observations,

w_i = weight assigned to observation i,

C = number of classes,

y_{i_c} = 1 if observation i belongs to class c, 0 otherwise,

p_{i_c} = probability estimated for observation i and class c.

For this research, weights are calculated as inverse proportions of class frequencies (King and Zeng (2001)), though it would also be reasonable to associate w_i with any meaningful business value, like credit exposure.

The important, yet sometimes confused point is that the log loss adapted for scoring purposes, similar to AUC or F1, needs to be distinguished from using this measure as a loss function optimized during algorithm training, e.g., in logistic regression or neural networks.

4. Data

Our analysis was conducted on the Lending Club data available online (https://www.lendingclub.com) for the period from June 2007 to December 2017, according to

the loan issuance date. The gathered dataset contains information about peer-to-peer lending activity with the status of the loan denoted as a response variable.

Since the predictions were to be made at the moment of credit application, we first filtered the data for the variables only known up to that time and thus prevented a potential data leakage. In the next step, all records with current loans were dropped (due to being non-informative for predictive model learning) and the remaining statuses were encoded as follows.

The *default* class was composed of charged off credits or borrowers that failed to repay for extended period of time. Loans that were due for 16-30 and 31-120 days were tagged as *late*. A 15-day grace period instances were excluded as it is hard to consider those actually late – many debtors extensively make use of an interest-free option. From business point of view, separation of the *late* class might be beneficial due to an early detection of potentially problematic customers. Fully paid loans fell into the last category. As a result, the dependent variable was constructed with *default*, *late* and *paid* classes, for which we also observed a major imbalance of 19.9%, 1.9% and 78.2%, respectively. After performing above operations and right before the train/test split, the dataset consisted of 1,223,196 observations and 47 raw explanatory variables available for further processing.

More in-depth investigation revealed several data problems. The pattern appeared for seven variables with the largest missing values percentage, as their missings were observed for the same period (approximately 3.7% samples). To our best knowledge, it seems like at the beginning the data for those variables has not been gathered. Yet, we decided not to discard affected records and subject them to a standard imputation process.

For many continuous predictors, a noticeable right skewness was also detected. This, for example, applies to income based features and is a natural reflection of a fact that individuals report the highest earnings. Skewed variables were handled by Yeo-Johnson transformation, shown in Appendix A. Among numerous categorical features, we identified a high cardinality ones, with multiple unique levels present and zip code being the most problematic (over 900 hundred distinct values). According to preprocessing scheme, those were addressed by target encoding (see Appendix B). Text variable, including a loan description provided by the borrower during application and composed of sentences of varying length and content, has been a subject to word2vec and k-means approach.

Finally, the aforementioned steps and data preprocessing allowed us to obtain a numerical matrix consisting of 175 columns for algorithm training.

5. Results

Results of our research are illustrated in Figure 4, which presents a log loss value obtained on 3-fold cross validation holdout samples (mean score), train and test datasets. All algorithms that were chosen to a final stacked ensemble are shown and the log loss is reported separately for the base and meta learners (the latter denoted with *meta* prefix). Classifiers are also sorted by their performance on the test set. The more precise numbers can be additionally found in Table 1.

Our analysis demonstrates two main findings. First and foremost, all meta models outperformed base learners, however the differences in relation to the best single base classifiers are not dramatic and might be of little practical business use, at least not as a priority. In particular, top three meta algorithms appeared to score very similar, with the slight advantage of the complex weighted learner. Second, we observe a significant share of well performing and efficient tree based methods. Both of these points are the subject of a further discussion.

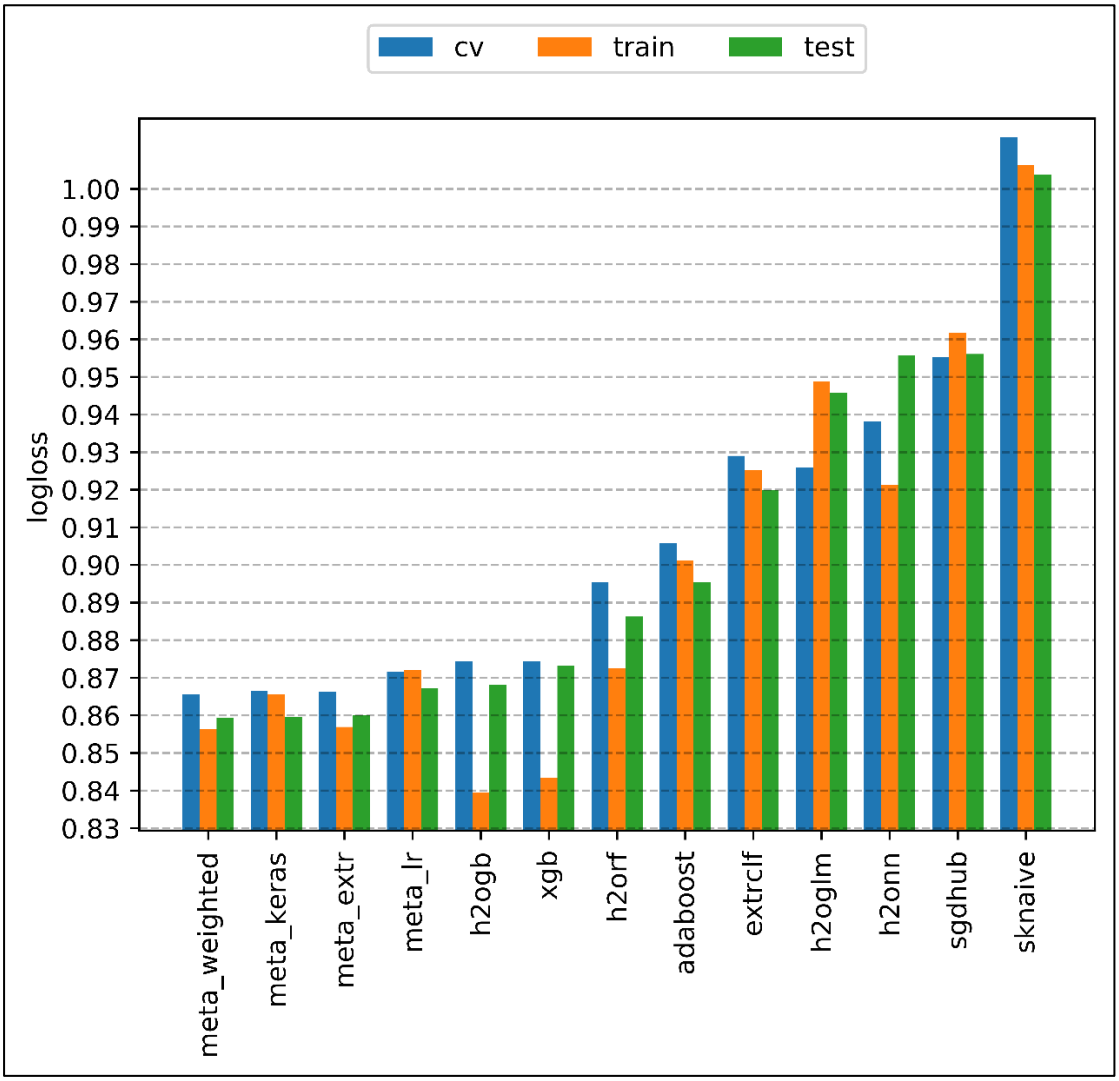

Figure 4: Results for base and meta learners, calculated on Ubuntu server with Python 3.7 and scikit-learn, h2o, xgboost and Keras libraries (last two GPU supported).

According to the inverse order in the Figure 4, nine base estimators used in the stacking solution are: Complement Naive Bayes, Stochastic Gradient Descent algorithm with modified Huber loss, feedforward neural network, regularized logistic regression, Extra-Trees classifier, AdaBoost, Random Forest, XGBoost and Gradient Boosting Machine. Other classifiers examined for this stage were discarded, either due to a large computational time burden (e.g., KNN, variations of SVM) or a poor performance compared to the *dummy* model that scored 1.94 for the test set (see Table 1).

In the second phase of ensemble learning, 27 new meta features allowed to train three meta learners and while a neural network (meta_keras) and logistic regression (meta_lr) were selected arbitrary, Extra-Trees estimator (meta_extr) provided more stable and slightly better results than XGBoost candidate, therefore being our final choice for the stacking architecture.

Lastly, the weighted meta model (meta_weighted) fitted in the third step was composed of $[0.0244, 0.4878, 0.4878]$ weights vector for meta_lr, meta_extr and meta_keras, respectively. Implications of introducing the weighted solution are also raised in the subsequent section.

Table 1: Log loss summary results for all learners used in the stacked ensemble

learners	cv	train	test
meta_weighted	**0.865503**	0.856358	**0.859343**
meta_keras	0.866597	0.865595	0.859534
meta_extr	0.866285	0.856912	0.860083
meta_lr	0.871553	0.872038	0.867112
h2ogb	0.874424	**0.839490**	0.868141
xgb	0.874410	0.843409	0.873250
h2orf	0.895326	0.872491	0.886271
adaboost	0.905681	0.901164	0.895343
extrclf	0.929000	0.925230	0.919961
h2oglm	0.925876	0.948772	0.945716
h2onn	0.938245	0.921203	0.955768
sgdhub	0.955162	0.961741	0.956091
sknaive	1.013754	1.006417	1.003702
dummy	—	1.943596	1.943596

Note: Bold numbers denote the best score obtained for a given type of the dataset: cross-validation holdout samples (cv), training set (train) and test set (test).

6. Discussion

In line with previous studies (among others, in Zhai and Chen (2018); Zhang et al. (2018); LeDell (2015)) we show that stacking improves the overall algorithm performance, although the complexity of the architecture presented and subtle gain boost may indicate applying it to those business areas that require enhanced results regardless of the costs or using it in the second place, after a well-established and simpler solution had been built. Minor differences in the achieved score are also of a great importance in the vast majority of machine learning competitions (van Veen et al. (2015)). To start with stacked ensembles, one can specifically consider applying a single meta learner instead of several algorithms or combinations, though we argue that using a simple classifier like logistic regression, as suggested for example by Whalen and Pandey (2013), might be the optimal selection. Indeed, according to our results, meta_lr was not able to fully capture complicated relationships. Choosing a more sophisticated

meta learner, e.g., from bagging or boosting family, might further elevate the performance, especially in case of nonlinearities presence among the meta features set.

A strong evidence for implementing tree based algorithms to obtain a good predictive results on tabular data is found, which confirms a conclusion reached by Olson et al. (2018) on the basis of 165 datasets. In our research, top 5 base learners were actually tree ensembles, with Extra-Trees estimator additionally being an important component of the meta learning architecture. What is more, satisfactory scores in aforementioned cases were achieved with a high efficiency due to the parallelization and GPU contribution. We also find a clear support for the thesis that any machine learning algorithm and tree ensembles in particular, can benefit substantially from meta features learning – Extra-Trees, an average base learner, performed incredibly well as a meta_extr classifier.

On the other hand, a slight tendency to overfit for this dataset was observed among tree boosting methods (very optimistic train score for h2ogb and xgb, see Figure 4), therefore indicating a necessity for a cautious and time consuming hyperparameter search. The latter directly refers to tuning the tree depth for Extra-Trees in meta_weighted solution, where the particular depth appeared to give satisfactory results while constantly overfitting in a single meta_extr model. This is in accordance with Shahhosseini et al. (2019) and proves that tuning meta hyperparameters altogether in a meta ensemble algorithm leads to a different estimates compared to treating meta learners separately.

The last point that should be discussed is the stability and generalizability of the stacking ensemble, which definitely requires a further investigation and we also consider it as a potential limitation of our research. However, to the extent that this can be determined by the presented study, we claim that the weighted meta learner, as an optimal combination of the best attributes of its components, is less prone to overfitting than any single algorithm examined. The weights vector recommended during the Bayesian search equally favored meta_extr and meta_keras classifiers while marginalizing a logistic regression. Thus, an analogy to the regularization can be made (and it is generally agreed that regularization reduces overfitting), where weights serve as coefficients and meta learners predictions are penalized, which might be additionally useful for meta algorithm selection process. Nevertheless, a thorough investigation, certainly including more than three meta learners and various datasets, would be appropriate for stronger evidence.

7. Conclusion

In this paper we propose a comprehensive, end-to-end stacked ensemble approach and evaluate its performance on the large, loan default dataset with multiple classes and high class imbalance presence – problems especially difficult to handle for single classifiers, yet widely faced in machine learning applications. Particularly, a complex multilayer structure consisting of diverse algorithms is built, with three separate meta learners examined and a combined, simultaneously tuned weighted ensemble fitted in the last stage. The whole architecture is also extended by a nested cross-validation schema for constructing unbiased meta features set. Results obtained indicate that stacking offers a noticeable improvement compared to individual estimators, although we conclude that any implementation should consider a relation between gains potentially acceptable only in certain business areas and increased computational load along with diminished model interpretability. The latter is especially important in the context of explainable AI (Arrieta et al. (2019)).

To overcome the imbalanced classification problem, weighted log loss metric is proposed, thus preventing models from being biased towards the majority class during training. Additionally, necessary preprocessing steps are outlined as an inherent part of an effective, not overfitted stacking solution. Specifically, we use Yeo-Johnson transformation in case of skewed variables and Random Forest multiple imputation technique for missing values. High cardinality categorical features are successfully addressed by a target encoding. Word2vec, K-means and KNN algorithms are utilized for a feature engineering or extraction processes.

In the second place, our findings confirm the supremacy of tree based methods for tabular data in predictive modeling. Not only their overall performance as base learners is satisfactory (including time efficiency), but also Extra-Trees algorithm appears to be a strong meta classifier, capturing more complicated data relationships than a popular logistic regression. Moreover, it is equally weighted with a neural network in the final ensemble. As a supplementary observation, cautious hyperparameter selection is recommended to prevent overfitting, in particular for bagged or boosted trees, and it is not guaranteed that the same hyperparameters set is an optimal choice in both separated and combined meta learning phase. For hyperparameter tuning purposes, Bayesian optimization is applied in this study.

Further research should aim to investigate the stability and generalizability of the presented stacking architecture, especially when implementing a weighted meta learner, which, in our opinion, is less prone to overfitting as it introduces regularization-like behavior to the system and might be potentially used for best meta algorithms selection.

References

Alaraj, M., and Abbod, M.F. 2016. "Classifiers consensus system approach for credit scoring." Knowledge-Based Systems 104: 89–105. https://doi.org/10.1016/j.knosys.2016.04.013.

Arrieta, A.B., D'iaz-Rodr'iguez, N., Ser, J.D., Bennetot, A., Tabik, S., Barbado, A., Garc'ia, S., Gil-L'opez, S., Molina, D., Benjamins, R., Chatila, R., and Herrera, F. 2019. "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI." Information Fusion 58: 82–115. https://doi.org/10.1016/j.inffus.2019.12.012.

Breiman, L. 1996. "Stacked Regressions." Machine Learning 24: 49–64. https://doi.org/10.1007/BF00117832.

Breiman, L. 2001. "Random Forests." Machine Learning 45: 5–32. https://doi.org/10.1023/A:1010933404324.

Brown, I., and Mues, C. 2012. "An experimental comparison of classification algorithms for imbalanced credit scoring data sets." Expert Systems with Applications 39 (3): 3446–3453. https://doi.org/10.1016/j.eswa.2011.09.033.

Büyükçakir, A., Bonab, H., and Can, F. 2018. "A Novel Online Stacked Ensemble for Multi-Label Stream Classification." Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18). New York, NY, USA: 1063–1072. https://doi.org/10.1145/3269206.3271774.

Cawley, Gavin C., and Talbot, Nicola L.C. 2010. "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation." The Journal of Machine Learning Research 11: 2079–2107.

Chen, T., and Guestrin, C. 2016. "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16: 785–794. https://doi.org/10.1145/2939672.2939785.

Džeroski, S., and Ženko, B. 2004. "Is Combining Classifiers with Stacking Better than Selecting the Best One?" Machine Learning 54 (3): 255–273. https://doi.org/10.1023/B:MACH.0000015881.36452.6e.

Ferri, C., Hernández-Orallo, J., and Modroiu, R. 2009. "An experimental comparison of performance measures for classification." Pattern Recognition Letters 30 (1): 27–38. https://doi.org/10.1016/j.patrec.2008.08.010.

Friedman, J., Hastie, T., and Tibshirani, R. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." Journal of statistical software 33 (1): 1–22. http://dx.doi.org/10.18637/jss.v033.i01.

Friedman, J.H. 2000. "Greedy Function Approximation: A Gradient Boosting Machine." Annals of Statistics 29 (5): 1189–1232. https://doi.org/10.1214/aos/1013203451.

Geron, A. 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media Inc.

Geurts, P., Ernst, D., and Wehenkel, L. 2006. "Extremely randomized trees." Machine Learning 63: 3–42. https://doi.org/10.1007/s10994-006-6226-1.

H2O.ai. n.d. "K-Means Clustering." Retrieved from http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/k-means.html.

Hashem, S. 1996. "Effects of Collinearity on Combining Neural Networks." Connection Science 8 (3-4): 315–336. https://doi.org/10.1080/095400996116794.

Hastie, T., Tibshirani, R., and Friedman, J. 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York, NY, USA: Springer New York Inc.

Hung, C., and Chen, J.-H. 2009. "A selective ensemble based on expected probabilities for bankruptcy prediction." Expert Systems with Applications 36 (3): 5297–5303. https://doi.org/10.1016/j.eswa.2008.06.068.

Irawan, F., and Samopa, F. 2019. "Accounts Receivable Seamless Prediction for Companies by Using Multiclass Data Mining Model." Proceedings of the 4th International Seminar on Science and Technology. http://dx.doi.org/10.12962/j23546026.y2019i1.5096.

Jeni, L. A., Cohn, J. F., and De La Torre, F. 2013. "Facing Imbalanced Data Recommendations for the Use of Performance Metrics." Proceedings of the International Conference on Affective Computing and Intelligent Interaction and workshops. ACII (Conference): 245–251. https://dx.doi.org/10.1109%2FACII.2013.47.

Kang, S., Cho, S., and Kang, P. 2015. "Multi-class classification via heterogeneous ensemble of one-class classifiers." Engineering Applications of Artificial Intelligence 43: 35–43. https://doi.org/10.1016/j.engappai.2015.04.003.

King, G., and Zeng, L. 2001. "Logistic Regression in Rare Events Data." Political Analysis 9 (2): 137–163. https://doi.org/10.1093/oxfordjournals.pan.a004868.

Krstajic, D., Buturovic, L.J., Leahy, D.E. et al. 2014. "Cross-validation pitfalls when selecting and assessing regression and classification models." Journal of Cheminformatics 6 (1): 10. https://doi.org/10.1186/1758-2946-6-10.

Kuhn, M., and Johnson, K. 2013. “Data Pre-processing.” In Applied predictive modeling: 30–35. New York, NY: Springer.

Kwon, J., Choi, K., and Suh, Y. 2013. “Double Ensemble Approaches to Predicting Firms’ Credit Rating.” Proceedings of PACIS 2013: 158. https://aisel.aisnet.org/pacis2013/158.

Lanes, M., Borges, E.N., and Galante, R.D. 2017. “The effects of classifiers diversity on the accuracy of stacking.” The 29th International Conference on Software Engineering and Knowledge Engineering: 323–328. https://doi.org/10.18293/SEKE2017-016.

LeDell, E. 2015. “Scalable Ensemble Learning and Computationally Efficient Variance Estimation.” University of California, Berkeley. https://escholarship.org/uc/item/3kb142r2.

Lessmann, S., Baesens, B., Seow, H.-V., and Thomas, L. 2015. “Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research.” European Journal of Operational Research. https://doi.org/10.1016/j.ejor.2015.05.030.

Lessmann, S., Haupt, J., Coussement, K., and De Bock, K. 2019. “Targeting customers for profit: An ensemble learning framework to support marketing decision-making.” Information Sciences. https://doi.org/10.1016/j.ins.2019.05.027.

Madasamy, K., Ramaswami, M. 2018. “Two-phase stacking ensemble to effectively handle data imbalances in classification problems.” International Journal of Advanced Research in Computer Science 9 (1): 908–913. https://doi.org/10.26483/ijarcs.v9i1.5495.

Marqués, A., García, V., and Sánchez, J.S. 2012. “Exploring the behaviour of base classifiers in credit scoring ensembles.” Expert Systems with Applications 39 (11): 10244–10250. https://doi.org/10.1016/j.eswa.2012.02.092.

Menahem, E., Rokach, L., and Elovici, Y. 2009. “Troika - An improved stacking schema for classification tasks.” Information Sciences 179 (24): 4097–4122. https://doi.org/10.1016/j.ins.2009.08.025.

Micci-Barreca, D. 2001. “A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems.” ACM SIGKDD Explorations Newsletter 3 (1): 27–32. https://doi.org/10.1145/507533.507538.

Mikolov, T., Chen, K., Corrado, G.S., and Dean, J. 2013. “Efficient Estimation of Word Representations in Vector Space.” CoRR abs/1301.3781. https://arxiv.org/abs/1301.3781.

Nanni, L., and Lumini, A. 2009. “An experimental comparison of ensemble of classifiers for bankruptcy prediction and credit scoring.” Expert Systems with Applications 36 (2, Part 2): 3028–3033. https://doi.org/10.1016/j.eswa.2008.01.018.

Nishio, M., Nishizawa, M., Sugiyama, O., Kojima, R., Yakami, M., Kuroda, T., and Togashi, K. 2018. "Computer-aided diagnosis of lung nodule using gradient tree boosting and Bayesian optimization." PloS One 13 (4): e0195875. https://dx.doi.org/10.1371%2Fjournal.pone.0195875.

Olson, R.S., Cava, W., Mustahsan, Z., Varik, A., and Moore, J.H. 2018. "Data-driven advice for applying machine learning to bioinformatics problems." Pacific Symposium on Biocomputing 23: 192–203. https://doi.org/10.1142/9789813235533_0018.

Pinto, D. 2017. "Feature Extraction with KNN." Retrieved from https://davpinto.github.io/fastknn/articles/knn-extraction.html.

Rasmussen, C. E., and Williams, C. K. I. 2006. Gaussian Processes for Machine Learning. The MIT Press.

Rennie, Jason D.M., Shih, L., Teevan, J., and Karger, David R. 2003. "Tackling the poor assumptions of naive bayes text classifiers." Proceedings of the Twentieth International Conference on Machine Learning (ICML'03): 616–623.

Shah, A.D., Bartlett, J.W., Carpenter, J., Nicholas, O., and Hemingway, H. 2014. "Comparison of random forest and parametric imputation models for imputing missing data using MICE: A CALIBER study." American Journal of Epidemiology 179 (6): 764–74. https://doi.org/10.1093/aje/kwt312.

Shahhosseini, M., Hu, G., and Pham, H. 2019. "Optimizing Ensemble Weights and Hyperparameters of Machine Learning Models for Regression Problems." ArXiv abs/1908.05287. https://arxiv.org/abs/1908.05287.

Sill, J., Takács, G., Mackey, L.W., and Lin, D. 2009. "Feature-Weighted Linear Stacking." ArXiv abs/0911.0460. https://arxiv.org/abs/0911.0460.

Sjardin, B., Massaron, L., and Boschetti, A. 2016. Large Scale Machine Learning with Python. Birmingham, England: Packt Publishing.

Snoek, J., Larochelle, H., and Adams, R. 2012. "Practical Bayesian optimization of machine learning algorithms." Advances in Neural Information Processing Systems 25: 2960–2968.

Tarap, K., and Stos, A. 2019. "Hit and Miss: An evaluation of imputation techniques from Machine Learning." Proceedings of the PhUSE US Connect 2019: Paper ML04.

Vahid, P.R., and Ahmadi, A. 2016. "Modeling corporate customers' credit risk considering the ensemble approaches in multiclass classification: evidence from Iranian corporate credits." Journal of Credit Risk 12 (3): 71–95. https://doi.org/10.21314/JCR.2016.213.

van der Laan, Mark J., Polley, Eric C., and Hubbard, Alan E. 2007. "Super Learner." U.C. Berkeley Division of Biostatistics Working Paper Series. Working Paper 222. https://doi.org/10.2202/1544-6115.1309.

van Veen, H.J., Nguyen, The Dat L., Segnini, A. 2015. "Kaggle Ensembling Guide." Retrieved from https://mlwave.com/kaggle-ensembling-guide.

Wainer, J., and Cawley, G.C. 2018. "Nested cross-validation when selecting classifiers is overzealous for most practical applications." ArXiv abs/1809.09446. https://arxiv.org/abs/1809.09446.

Wan, S., and Yang, H. 2013. "Comparison among Methods of Ensemble Learning." Proceedings of the International Symposium on Biometrics and Security Technologies, ISBAST 2013: 286–290. https://dx.doi.org/10.1109/ISBAST.2013.50.

Wang, H., and Castanon, J.A. 2015. "Sentiment expression via emoticons on social media." Proceedings of the 2015 IEEE International Conference on Big Data: 2404–2408. https://doi.org/10.1109/BigData.2015.7364034.

Whalen, S., and Pandey, G. 2013. "A Comparative Analysis of Ensemble Classifiers: Case Studies in Genomics." Proceedings of the 2013 IEEE 13th International Conference on Data Mining: 807–816. https://doi.org/10.1109/ICDM.2013.21.

Wolpert, D.H. 1992. "Stacked Generalization." Neural Networks 5 (2): 241–259. https://doi.org/10.1016/S0893-6080(05)80023-1.

Wu, T., Lin, C., and Weng, R.C. 2004. "Probability Estimates for Multi-class Classification by Pairwise Coupling." Journal of Machine Learning Research 5: 975–1005. https://dl.acm.org/doi/10.5555/1005332.1016791.

Xiao, H., Xiao, Z., and Wang, Y. 2016. "Ensemble classification based on supervised clustering for credit scoring." Applied Soft Computing 43: 73–86. https://doi.org/10.1016/j.asoc.2016.02.022.

Yamashita, T., Sato, N., Kino, H., Miyake, T., Tsuda, K., and Oguchi, T. 2018. "Crystal structure prediction accelerated by Bayesian optimization." Physical Review Materials 2 (1): 013803. https://doi.org/10.1103/PhysRevMaterials.2.013803.

Yeo, I., and Johnson, R. 2000. "A New Family of Power Transformations to Improve Normality or Symmetry." Biometrika 87 (4): 954–959. https://doi.org/10.1093/biomet/87.4.954.

Zdravevski, E., Lameski, P., Kulakov, A., and Kalajdziski, S. 2015. "Transformation of nominal features into numeric in supervised multi-class problems based on the weight of evidence parameter." Proceedings of the 2015 Federated Conference on Computer

Science and Information Systems (FedCSIS): 169–179. http://dx.doi.org/10.15439/2015F90.

Zhai, B., and Chen, J. 2018. "Development of a stacked ensemble model for forecasting and analyzing daily average PM2.5 concentrations in Beijing, China." Science of the Total Environment 635: 644–658. https://doi.org/10.1016/j.scitotenv.2018.04.040.

Zhang, Y., Liu, G., Luan, W., Yan, C., and Jiang, C. 2018. "An approach to class imbalance problem based on stacking and inverse random under sampling methods." Proceedings of the 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC): 1–6. https://doi.org/10.1109/ICNSC.2018.8361344.

Zhang, Z. 2015. "Missing data exploration: highlighting graphical presentation of missing pattern." Annals of translational medicine 3 (22): 356. https://doi.org/10.3978/j.issn.2305-5839.2015.12.28.

Zhu, J., Zou, H., Rosset, S., and Hastie, T. 2009. "Multi-class AdaBoost." Statistics and its interface 2 (3): 349–360. https://doi.org/10.4310/SII.2009.v2.n3.a8.

Appendix A

Yeo-Johnson transformation for continuous variables

Figure A1: Variables before (left picture) and after (right picture) Yeo-Johnson power transformation. Distributions on the right appear to have more Gaussian-like distribution, though skewness is still persistent in several cases. Response classes are also distinguished.

Appendix B

Target encoding of high cardinality variables

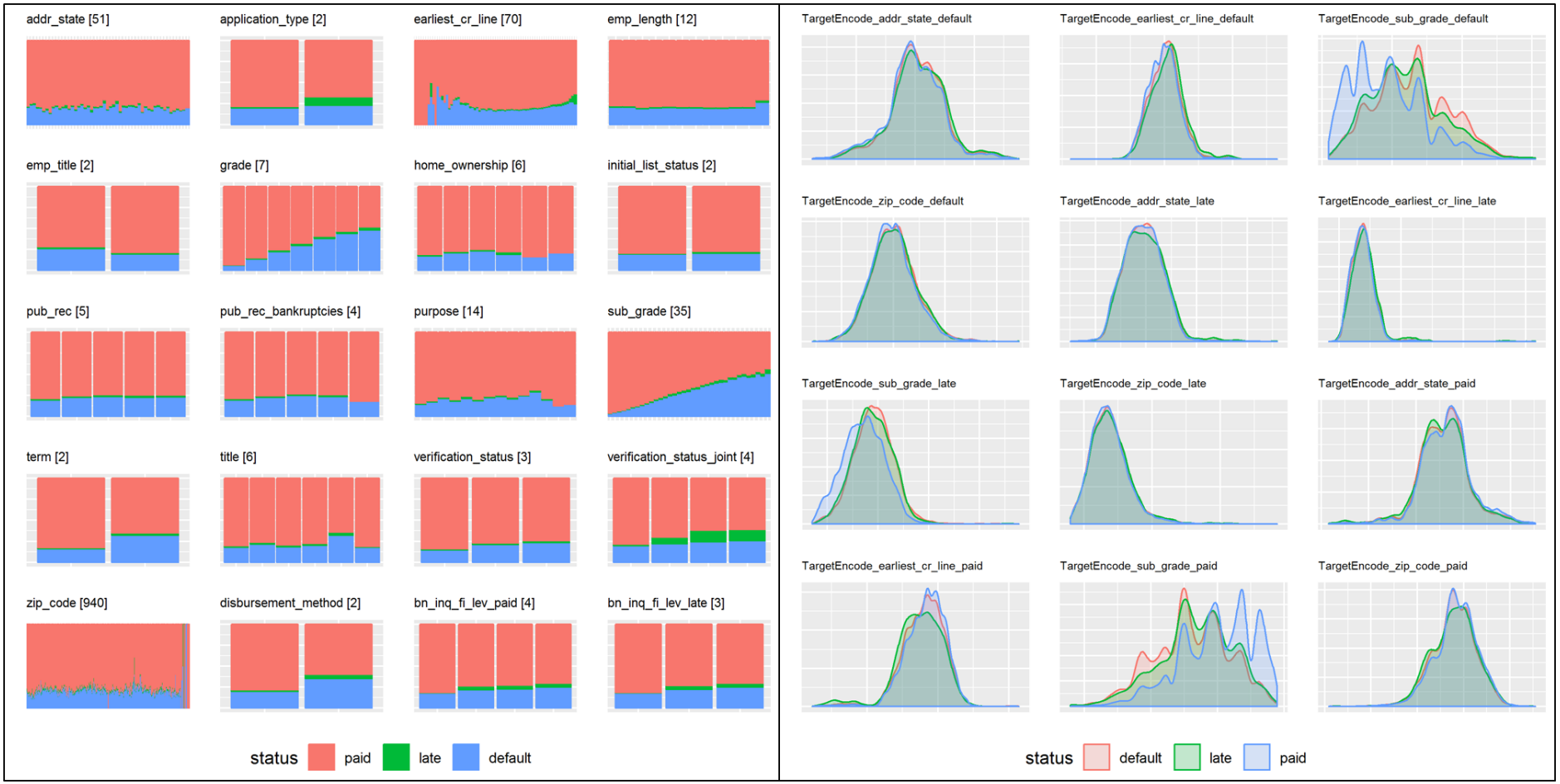

Figure B1: Bar plots for all categorical variables (number of levels in square brackets) and target encoding for the most high cardinal predictors (right picture), in one versus all manner. Twelve continuous features are generated instead of utilizing hundreds of unique categorical levels.

University of Warsaw
Faculty of Economic Sciences
44/50 Długa St.
00-241 Warsaw
www.wne.uw.edu.pl