

HUMBOLDT UNIVERSITY OF BERLIN

COURSE: NUMERICAL INTRODUCTORY SEMINAR

SUPERVISOR: PROF. DR. BRENDA LÓPEZ CABRERA



Stacking Algorithm and Ensemble Modelling

Submitted by

FREDERIK SCHRECK

STATISTICS M.SC.

HUMBOLDT UNIVERSITY

580567

July 14, 2018

Contents

1	Motivation	1
2	Stacking and Ensembling Modelling	2
2.1	Bagging and the Random Forest model	3
2.2	Boosting and the Gradient Boosting model	4
2.3	Stacked Generalization model	5
3	Credit Risk Assessment	6
4	Methodology	7
4.1	Data description	7
4.2	Model building process	7
4.3	Evaluation metrics	8
5	Results	10
6	Conclusion	13
A	Appendix - Summary Tables	14
B	Appendix - Code	16

List of Figures

1	Bias and Variance Reduction	6
2	Correlation of predictions	11
3	ROC curves	12

List of Tables

1	Model performances on the test dataset	10
2	Summary statistics for numerical features	14
3	Summary statistics for categorical features	15

List of Quantlets

1	Quantlet 1: Main file	16
2	Quantlet 2: Gradient Boosting Model Feature Selection	25
3	Quantlet 3: Random Forest Model Feature Selection	26
4	Quantlet 4: Stacking: Decision Tree (level 0) Feature Selection	27
5	Quantlet 5: Logistic Regression (level 0) Feature Selection	27
6	Quantlet 6: Stacking: Neural Network (level 0) Feature Selection	28
7	Quantlet 7: Gradient Boosting (level 0) Feature Selection	29
8	Quantlet 8: Stacking: Random Forest (level 0) Feature Selection	30
9	Quantlet 9: Random Forest Model Building	30
10	Quantlet 10: Gradient Boosting Model Building	32
11	Quantlet 11: Stacking: Decision Tree (level 0) Model Building	33
12	Quantlet 12: Stacking: Logistic Regression (level 0) Model Building	35
13	Quantlet 13: Stacking: Neural Network (level 0) Model Building	36
14	Quantlet 14: Stacking: Gradient Boosting (level 0) Model Building	37
15	Quantlet 15: Stacking: Random Forest (level 0) Model Building	39
16	Quantlet 16: Stacking: Stacked Generalization Model Building	41

Stacking Algorithm and Ensemble Modelling

Frederik Schreck

July 14, 2018

Abstract

Stacking and Ensemble models currently belong to the most powerful machine learning tools. In depth, this paper introduces and discusses the most important concepts of Stacking and Ensembling. In order to assess their predictive performance in the context of credit risk assessment, a empirical evaluation study is realized on behalf of the German Credit Dataset. Results show that Ensembling models, including a Random Forest and a Gradient Boosting model, outperform standard machine learning models on a broad set of evaluation metrics. Different versions of Stacked Generalization models were able to establish better predictions than their level 0 generalizers. Thereby, more sophisticated Stacking algorithms could establish even better results. The restriction of the subset of input predictions for the Stacked Generalization models was ineffective with regards to performance issues. The results strongly reinforce the value of Stacking and Ensembling strategies for prediction in credit risk assessment problems.

1 Motivation

"[W]hen our imperfect judgements are aggregated in the right way, our collective intelligence is often excellent." (Surowiecki, 2005, Foreword p.XIV)

In accordance with the title of his book, Surowiecki refers to what he calls the *wisdom of crowds*-phenomenon (Surowiecki, 2005). This social phenomenon describes that - under certain fulfilled criteria - the aggregates of individual judgements are superior to each individual judgement alone. While this effect can be found in the social world, it also applies to the world of statistics and machine learning. In the field of Stacking and Ensemble modelling, research has shown different ways in which the aggregation of predictive models can deliver a more powerful model. Such Stacking and Ensemble models

currently belong to the most powerful machine learning tools as can be seen by their success in data science competitions (e.g. on the data science competition website Kaggle). In the context of credit risk assessment, where banks need to estimate credit worthiness of potential customers, accurate predictions are particularly valuable. In comparison to alternative predictive models, Stacking and Ensemble methods have shown to be highly effective in this area (Yu et al., 2008; Zhu et al., 2017). The field of existing methods is however large. In depth, this paper will therefore introduce and apply the most important concepts, namely Bagging and the related Random Forest model (Breiman, 1996, 2001), the concept of boosting and the related Gradient Boosting model Freund and Schapire (1996); Friedman (2002) as well as the idea of Stacked Generalization (Wolpert, 1992).

The paper’s structure is as follows: In the coming section, the different concepts of Stacking and Ensembling shall be introduced and their strengths and shortfalls will be discussed. In the following, the value of applying these models in the context of credit risk classification is discussed briefly. Subsequently, an empirical evaluation study that aims at applying the introduced Stacking and Ensemble models in such financial context is prepared. For that, firstly the credit risk data is presented, secondly the model building process is outlined and thirdly the metrics for model evaluation are introduced. In the next section, results of the empirical evaluation study are presented in detail. Finally, conclusions about comparative advantages and shortfalls of the models in the context of credit risk classification are drawn and needs of further research are identified.

2 Stacking and Ensembling Modelling

Ensemble learning generally refers to the combination of multiple hypotheses in order to obtain a more powerful hypothesis. In the context of machine learning, the term *hypothesis* refers to the output of an algorithm, which aims to learn a target function $f(\mathbf{x})$ by using a set the features \mathbf{x} . Each algorithm that is used in the combination process of an ensemble learner is called a *base learner*.

Formally, this means that given training data $D^{train} := \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, where $x_i = (x_{i1}, x_{i2}, \dots, x_{iK})$ is the vector of feature values for each observational unit $i \in \{1, 2, \dots, N\}$, $y = f(\mathbf{x})$ is the target vector that can be modelled by an unknown function f of the features. Further let K denote the number of features and N denote the number of observations. Then, a set of base learners of size M delivers hypotheses $h_1, h_2, \dots, h_M \in H$ about the true function f , whereby H denotes the hypothesis space.

Different ways to combine the hypotheses of base learners exist. Generally, ensemble learning is most effective when diverse base learners are combined. Hereby, diversity refers to error diversity, implying that the different base learners have different strengths in capturing structure in the data. Brown et al. (2005) show that the combination methods of

ensemble learning strategies enhance such diversity. Furthermore, practical applications of ensembling techniques show, that Stacking and Ensemble models enhance diversity amongst their base learners by requiring different algorithms, different hyperparameter settings, different feature subsets and different training sets for their base learners (Güneş, 2017).

In the following, different techniques to combine the hypotheses of base learners will be introduced and their specific strengths and shortfalls are identified.

2.1 Bagging and the Random Forest model

The idea of Bagging was originally proposed by Breiman (1996). The term abbreviates bootstrap aggregating which refers to a manipulation of the training data: Each base learner m is fitted on a uniformly drawn random sample D_m^{train} from the training data D^{train} . Notably, D_m^{train} may contain duplicates of certain observational units, since sampling is done with replacement. The hypotheses of the base learners is then aggregated by averaging in case of a regression problem or majority voting in case of a classification problem.

In so far, bagging is a meta-algorithm that can be used with every type of base learner algorithm. However, especially unstable base learners that are sensitive to data manipulation should be combined (Breiman, 1996, p.124). Breiman therefore recommends to use Neural Networks, Decision Trees or subset selection in Linear Regression. By building the base learners on different subsets of the data, the bagging procedure enhances diversity amongst them and can lead to "substantial gains in accuracy" (Breiman, 1996, p.123).

The Random Forest model uses the bagging principle and supplements it with the random subspace approach (Ho, 1998; Breiman, 2001). This approach builds each base learner on a random sample with replacement of all available features, which implies a decorrelation of the base learners' hypotheses. For Random Forests, the Decision Trees is the preferred base learner algorithm (cp. Breiman, 2001).

A big strength of the Random Forest is the reduction in prediction variance compared to single Decision Trees, which stems from the diversification. Clearly, the computational costs of a Random Forest can be much higher than those of a single Tree, since computational time increases linearly to the number of consulted Trees. Due to the independent building of the individual base learners, Random Forest model building can be accelerated by parallelization on different cores of the computer. Breiman further notes that Random Forests give "useful internal estimates of error, strength, correlation and variable importance" (Breiman, 2001, p.10). Hence, even though decision rules of Random Forests are less transparent than those of single Decision Trees due to the sampling, relative variable importances can be calculated as a by-product by randomly permuting

features and examining their influence on the prediction. As a single weakness, growing a large random forest can be computationally expensive. Breiman proves that the generalization error of the ensemble converges almost surely to a limit with increasing number of Trees (Breiman, 2001, p.30). In practice, the number of Trees is however restricted by the amount of available computational resources.

2.2 Boosting and the Gradient Boosting model

Beneath Bagging, another powerful ensembling technique is Boosting. It builds on the idea that the aggregation of weak base learners may lead to a strong learner. In their *Adaboost* algorithm, Freund and Shapire (1996) start with an ensemble of one weak learner and iteratively add one more weak learner that aims to correct for the (pseudo) residuals of the current ensemble. Thereby, the calculation of these residuals is based on an iterative reweighting of the data. The weights of each datapoint x_i for model m depend on the prediction accuracy of the current ensemble hypothesis for that datapoint $h_{\{1,2,\dots,m-1\}}^{ensemble}(x_i)$.

With Boosting, it is possible to decrease the training error to zero (Freund and Schapire, 1996, p.11ff.). Furthermore, as long as base learners are better than random guessing, the Boosting technique is also able to reduce the generalization error independent of the base learning algorithm.

A development of the Boosting idea is the (Stochastic) Gradient Boosting model, which is currently the most commonly used Boosting model (Friedman, 2001, 2002). In contrast to the data weighting scheme in the Adaboost algorithm, Gradient Boosting minimizes the gradient of a loss function of the error by applying gradient descent. Typically, small Decision Trees are used as base learners of the Gradient Boosting model due to their propensity towards high prediction bias. Additionally, Gradient Boosting integrates the bagging idea. Friedman shows, that this integration could substantially improve accuracy and execution speed of the model (Friedman, 2002).

The possibility to be executed fast is a reason for the heavy use of Gradient Boosting models for machine learning problems. Furthermore, they allow to gain insights into the dependence of target and features by enabling partial dependence plots (Friedman, 2001, p.1219ff.). However, due to its nature, Gradient Boosting models are highly prone to overfit the training data and therefore must be accompanied with regularization methods (Friedman, 2002, p.1203). Therefore most of the parameters of the Gradient Boosting model deal with this problem of overfitting.

2.3 Stacked Generalization model

Stacked Generalization models has been introduced by Wolpert (1992) and defines a way to combine multiple predictive algorithms by using a second-level algorithm. In contrast to Bagging or Boosting, Stacked Generalization is typically applied to a space of different base learner algorithms. The idea is, that different kinds of models that are applied to the learning problem are able to capture only part of the problem. Combining models with diverging strengths in the right way then leads to improved predictive accuracy. Stacked Generalization is therefore also referred to as a second-stage model.

The Stacking algorithm involves partitioning of training dataset $D^{train} = (\mathbf{x}^{train}, y^{train})$ into $J = \{1, 2, \dots, J\}$ disjoint parts $D_1^{train}, D_2^{train}, \dots, D_J^{train}$. For each of these subsets D_j^{train} , called *level 0 learning set*, a base learner $m \in \{1, 2, \dots, M\}$, also referred to as *level 0 generalizer*, is built on behalf of the training dataset D_{-j}^{train} that does not include subset j . (Wolpert, 1992, cp.). In each iteration, the model built on D_{-j}^{train} is used to predict the target feature in subset D_j^{train} . The predictions of the J subsets are then combined again in order to obtain a prediction of the target over the whole training dataset D^{train} . Besides that, each level 0 generalizer is used to predict on the test dataset D^{test} . Due to the level 0 generalizers being built J times on different disjoint subsets of the training data, Stacked Generalization can be seen as a sophisticated form of cross-validation. The next step is building a meta learner, referred to as *level 1 generalizer*, that produces a prediction by using the training dataset predictions of the M level 0 generalizers as inputs.

Wolpert shows that this stacking procedure is able to reduce the bias of the base learners and thus minimizes the generalization error rate. He even recommends to use a version of Stacked Generalization in any real-world problem (Wolpert, 1992, p.2).

Different meta learning algorithms can be used for the combination of base learners. An optimal meta algorithm finds the best way to use the strengths of the base learners. Overfitting problem is especially present in Stacked Generalization models. This is due to the base learners all predicting the same target (Güneş, 2017). As a consequence, cross-validation and regularization can be used. Further more, the chosen meta learning algorithm should not be sensible to collinearity. It is therefore especially recommended to use Regularized Regression, Gradient Boosting or hill climbing methods (Güneş et al., 2017).

Figure 1 visualizes how Boosting, Bagging and Stacking are able to increase performance of predictive models by decreasing generalization variance and generalization bias of Decision Tree base learners, exemplarily. Furthermore, it can be seen that making the model more complex, by e.g. growing the Tree, does typically not provide a good model.

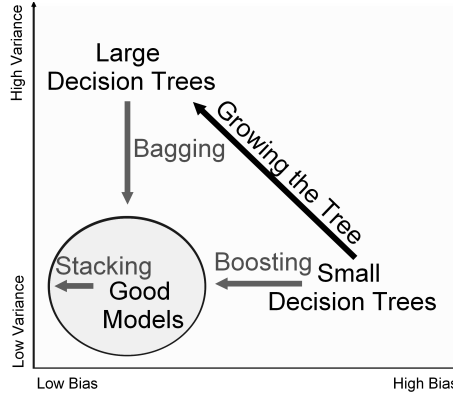


Figure 1: Reduction of Bias and Variance by Stacking and Ensembling methods with Decision Trees as base learners.

3 Credit Risk Assessment

Techniques of Stacking and Ensemble modelling are applied to predictive problems of a broad range of topics. This paper will especially focus on the application of Ensemble Learning in credit risk classification problems. Credit risk assessment and especially its modelling is an important part in the field of financial risk management since for most small- and medium-sized banks, interests on loans are still the primary financial source (Jacobson et al., 2006, p.2). The banking supervision accord Basel II, that was published in 2004 and applies to member states of the European Union since 2007, restricted the buffer capital on banks and therefore makes it especially important for them to estimate the riskiness of loan applicants (Basel Committee on Banking Supervision, 2004). For that, banks need to be able to distinguish between risky and non-risky applicants. Two opposing factors determine the banks' business rules regarding loans: On the one hand, more loans are better for the bank. On the other hand, a bank can not afford to make too many bad decisions, since this would eventually lead to a collapse of the bank. A good strategy on granting loans will therefore be a compromise. Minimizing the fraction of bad decisions thus increases banks' revenue.

Applying Ensemble Learning techniques to credit risk modelling has already proven to be highly valuable. Zhu et al. (2017) investigate credit risk assessment for small- and medium-sized Chinese enterprises. For that, they carry out an experiment in which they compare the predictive performance of individual machine learning methods and Ensembling methods of different complexity. They find especially the more complex ensembling methods to be of outstanding discriminative accuracy (Zhu et al., 2017, p.46f.). Yu et al. (2008) successfully apply an Ensemble learner comprising six levels of stacked Neural Networks in order to evaluate credit risk at the measurement level. Hereby, they further incorporate the Bagging approach. They conclude, that such technique "provides a promising solution to credit risk analysis" (Yu et al., 2008, p.1443).

4 Methodology

In order to evaluate and compare the introduced Stacking and Ensemble models, an empirical evaluation study is conducted. The quantlets for replication of the study can be found in the Appendix B or accessed in the corresponding github repository. In this section, the dataset used for the evaluation study is presented, the model building process is explained in detail and the metrics for evaluation are introduced.

4.1 Data description

The empirical evaluation study of this paper uses the German Credit Dataset from the UCI machine learning repository (Dheeru and Karra Taniskidou, 2017). This dataset classifies people as either being good or bad customers for a bank with respect to credit risk. It comprises a total number of 1000 observations and 20 features. Tables 2 and 3 in the Appendix present the summary statistics for the numerical and the categorical features in the original dataset, respectively. To ensure better model performance, the numerical data is standardized before model building. The dataset is partitioned into a training dataset, comprising 750 observations, and into a test dataset, comprising 250 observations.

4.2 Model building process

The model building process consists of feature selection, model training and tuning as well as prediction. For the purpose of this study, an extensive set of models went through this process: a Random Forest model and a Gradient Boosting model represent the Ensemble learners. Furthermore, both models are built a second time as level 0 generalizers for the Stacked Generalization models. Additionally, a Decision Tree, a Neural Network as well as a Logistic Regression model are built in order to provide a diverse set of level 0 generalizers for the Stacking. Four different such Stacked Generalization models are built by using different subsets of base learners' predictions and different combiner algorithms.

Before training the models, feature selection is a critical step. The aim of feature selection is dimension reduction. Building the models on an optimal subset of features may reduce their training time, reduce the variance and may make the model more easily interpretable (Guyon and Elisseeff, 2003). Since, the optimal subset of features depends on each model, a wrapper approach for feature selection is applied to each model specifically. Each model-specific wrapper approach starts with building an intercept model and sequentially adding the next best feature by using a sequential forward selection approach. The wrapper approach stops when adding another feature cannot increase the AUC measure (see section 4.3) by at least 0.00001 units. All wrapper approaches are run

on 3-fold cross-validation in order to avoid overfitting problems. Notably, a larger number of folds leads to instabilities due to the small sample size. Subsequently, the subset of features identified by the model-specific wrapper approaches is used for training of the corresponding models. Since the Random Forest and the Gradient Boosting models are built as Ensemble models as well as level 0 generalizers for the Stacked Generalization models, independent wrapper approaches are applied for both versions.

The training process for each model generally consists of establishing a broad-grid tuning of all relevant hyperparameters on the training dataset in order to find the (locally) optimal parameter choices. In order to avoid overfitting on the training dataset, each combination of hyperparameters is tested by a 3-fold cross-validation process. For the Random Forest and the Gradient Boosting model, their tuned versions can directly be used for prediction on the test dataset. For the Stacked Generalization models, the training dataset is partitioned into five disjoint subsets. Each of the five level 0 generalizers is then built in five iterations as described in section 2.3. Again, each iteration involves a parameter tuning on 3-fold cross-validation. All level 0 generalizers are then used to predict the observations in the test dataset. Stacking model 1 is then built by averaging over the probabilistic predictions of all five level 0 generalizers. Stacking model 2 is constructed by averaging over the probabilistic predictions of the three best predictions of level 0 generalizers in terms of AUC measure. For that, the correlations of the level 0 generalizers on the training data must be investigated in order to avoid multicollinearity problems. Stacking model 3 is built by using again all level 0 predictions and a Gradient Boosting model as combining algorithm. Finally, Stacking model 4 is constructed by combining all level 0 predictions by using a Logistic Regression combiner. The parameters of the combining algorithms are again tuned under 3-fold cross-validation.

4.3 Evaluation metrics

In credit risk modelling, the misclassification costs are often type-specific. This means that a false negative prediction may have different costs for the bank than a false positive prediction. When misclassification costs are known, a cost-sensitive model building strategy should be consulted. Since in the context of this paper misclassification costs are however unknown, equal misclassification costs for false negative and false positive predictions are assumed. Furthermore, the broad field of cost-sensitive learning may serve as a topic for other studies. The following metrics are used to evaluate the models.

AUC: In the model building process, tuning of model parameters for each model m is evaluated on the Area Under Curve (AUC) metric. Tuning on the AUC is especially recommended when facing probabilistic predictions, since the metric generalizes over all

possible cut-off thresholds that could be used to transform probabilistic into binary predictions. The AUC is a ranking indicator that measures the area under the receiver operating characteristic curve (ROC curve) (Hanley and McNeil, 1982). For a probabilistic prediction, like in the context of this study, a visualization of the ROC curve can be obtained by plotting the sensitivity against $1 - \text{specificity}$ for all cut-off thresholds between zero and one. The AUC value can therefore take values between zero and one as well. A random model would obtain an AUC value of 0.5, which can thus function as a benchmark value in model evaluation on AUC. In a statistical sense, the AUC estimates the probability that a randomly chosen correct prediction is correctly ranked higher than a randomly chosen false prediction. For model m , the AUC can be calculated as

$$\text{AUC}^m = \frac{1}{P \times N} \sum_{j=1}^P \sum_{k=1}^N (\hat{y}_j^m - \hat{y}_k^m) \quad (1)$$

, whereby P and N denote the positive (in our case *good*) and negative (*bad*) instances amongst the outcome values in the credit data. Further, \hat{y}_j^m and \hat{y}_k^m are the predictions of model m for the positive instance y_j and the prediction for the negative instance y_k , respectively.

Accuracy: Another important metric in evaluation of classification models is the Accuracy metric, which can be interpreted as the percentage of correctly classified points. In contrast to the AUC, probabilistic predictions must be transformed into binary predictions for the Accuracy metric, which implies selecting a cut-off threshold. For the purpose of this study, a natural cut-off threshold of 0.5 is chosen.

$$\text{Accuracy}^m = \frac{TP^m + TN^m}{FP^m + FN^m + TP^m + TN^m} \quad (2)$$

, whereby TP^m is the number of true positive predictions for model m and TN^m , FP^m and FN^m are the corresponding number of true negatives, the number of false positives and the number of false negatives for model m , respectively.

Logarithmic Loss: The Logarithmic Loss is a metric for evaluating class predictions that penalizes for a high confidence about incorrect classifications. For the case of a binary outcome, the Logarithmic Loss is given by

$$\text{LogLoss}^m = -\frac{1}{N} \sum_{i=1}^N (y_i \times \log(p_i^m) + (1 - y_i) \times \log(1 - p_i^m)) \quad (3)$$

, whereby p_i^m is model m 's prediction for observation y_i .

Brier Score: The models will further be assessed on the Brier Score, which is identical to the Mean Squared Error metric in statistics. For model m , the Brier score is defined as

$$\text{Brier}^m = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i^m)^2 \quad (4)$$

, whereby \hat{y}_i^m denotes the predicted probability of model m for observation y_i .

5 Results

In the following, the results of the empirical application of Stacking and Ensemble models on a credit risk classification problem will be presented. Table 1 shows the evaluations of all models on the test dataset with respect to the metrics AUC, Accuracy, Logarithmic Loss and Brier Score.

	AUC	Accuracy	LogLoss	Brier
Ensemble model: Random Forest	0.78	0.73	0.52	0.18
Ensemble model: Gradient Boosting	0.78	0.72	0.52	0.18
Level 0 generalizer: Decision Tree	0.69	0.65	0.60	0.21
Level 0 generalizer: Logit Regression	0.71	0.74	0.57	0.19
Level 0 generalizer: Neural Network	0.76	0.74	0.53	0.18
Level 0 generalizer: Random Forest	0.73	0.69	0.56	0.19
Level 0 generalizer: Gradient Boosting	0.80	0.74	0.52	0.17
Stacking model 1: Average, all predictions	0.77	0.64	0.24	0.18
Stacking model 2: Average, best predictions	0.77	0.63	0.25	0.18
Stacking model 3: GB, all predictions	0.78	0.61	0.29	0.17
Stacking model 4: LR, all predictions	0.81	0.64	0.30	0.16

Table 1: Model performances on the test dataset. Values rounded on two digits after comma. Abbreviations GB and LR denote Gradient Boosting and Logistic Regression that are used as combiner algorithms.

Since all models were tuned on the AUC metric, the comparison on behalf of this metric is most informative. The two Ensemble models, namely the Random Forest and the Gradient Boosting model, score highly on the AUC metric with a value of 0.78. In terms of AUC, they even outperform most level 0 generalizers, especially the Decision Tree and the Logistic Regression. It can be concluded that they rank a randomly chosen correct prediction comparatively higher than a randomly chosen false prediction in comparison with the level 0 generalizers. Notably, the Gradient Boosting level 0 generalizer and the Random Forest level 0 generalizer perform a bit different when compared to their counterparts. This may origin from the decreased size of training dataset due to the partitioning that was applied for building the level 0 generalizers.

The four Stacking models were built on top of the level 0 generalizers. Figure 2 shows the correlations of the predictions of the level 0 generalizers on the training data. It can be seen that the predictions of the level 0 generalizers are (mostly) correlated positively, which seems intuitive since they all predict the target in the similar context. However, correlations are not perfect (all $< |0.9|$). This reveals diversity of predictions, which already indicates that combining them may increase predictive performance.

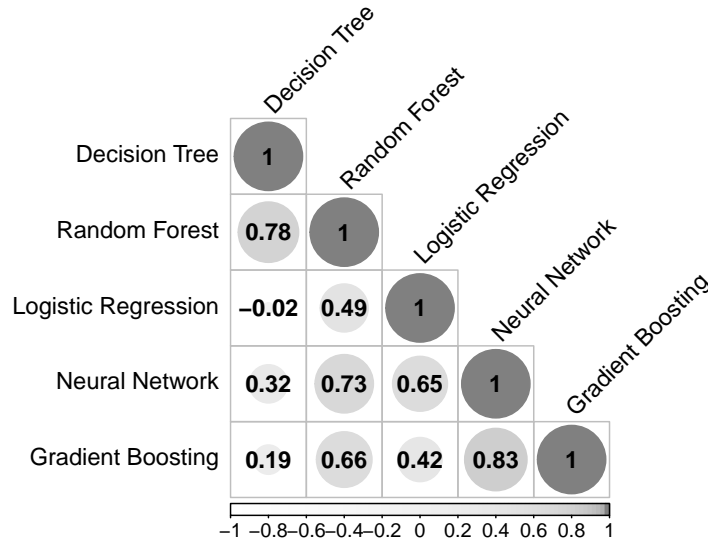


Figure 2: Correlation plot of training dataset predictions of level 0 generalizers.

Indeed, Table 1 reveals that the four Stacking models clearly outperform their level 0 generalizers in terms of AUC. Stacking models 1, 2 and 3 that are based on averaging predictions as well as a Gradient Boosting combiner have similar high AUC performance than the Random Forest and the Gradient Boosting model. Stacking model 4 that is based on combining level 0 predictions by Logistic Regression even shows a better AUC performance than the Random Forest and the Gradient Boosting model. For visual comparison, Figure 3 shows the ROC curves related to the AUC values for all models.

With regards to the Accuracy, the Logistic Regression and the Neural Network perform about equally well as the two Ensemble models. The Decision Tree is however outperformed by the Random Forest and the Gradient Boosting model. Interestingly, all four Stacking models show relatively bad values on the Accuracy metric. Regarding the relation of Accuracy and AUC, it can be concluded that the Stacking and Ensemble models do not necessarily perform better on the cut-off threshold of 0.5, which is implied by the Accuracy metric. However, generalizing over all possible cut-off threshold they clearly outperform most level 0 generalizers as can be seen by the corresponding AUC values and the corresponding ROC curves in Figure 3. Furthermore, this indicates that the standard cut-off threshold for probabilistic predictions of 0.5 may not be optimal for those models.

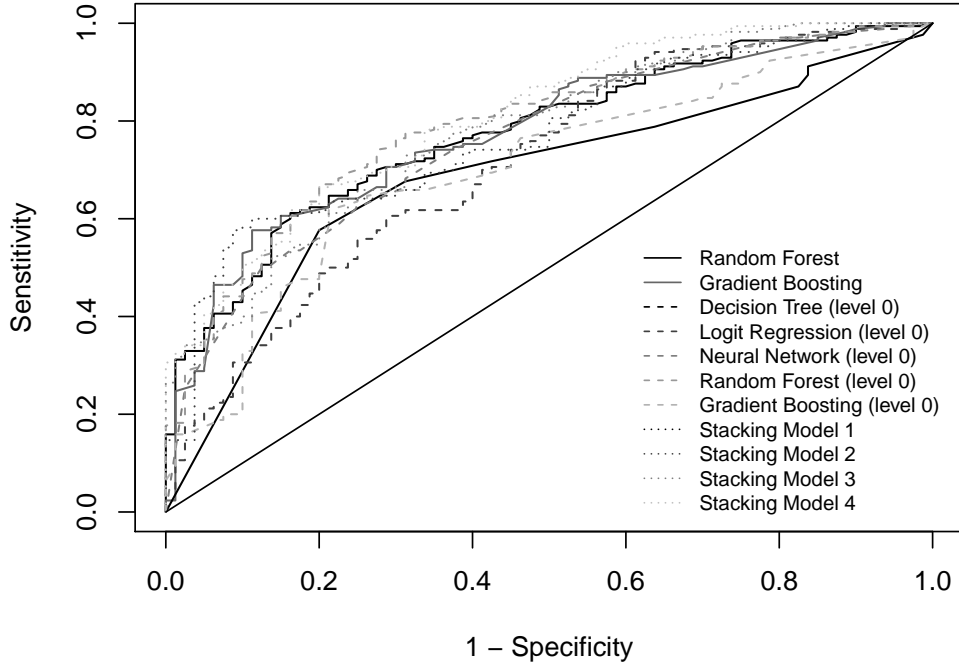


Figure 3: Receiver Operating Characteristic (ROC) curves for the predictions of all models. The diagonal line represents the ROC curve of a random model.

The Logarithmic Loss metric gives even more information on the particular strengths and weaknesses of the models. Contrasting the level 0 generalizers with the Random Forest and the Gradient Boosting model, the latter show a slightly smaller Logarithmic Loss. The four Stacking models however, reveal a much better performance on behalf of the Logarithmic Loss metric. It can be concluded, that the Ensemble models and even more the Stacking models do not place as high confidences on their incorrect predictions.

Regarding the Brier Score, differences are not that present. In tendency, the Stacking and Ensemble Models perform slightly better on that metric than the level 0 generalizers. One time more, this shows that in binary classification a good model is not only defined by its errors but rather by which observations it is able to classify correctly.

With regards to all calculated metrics, Stacking models are able to capture the structure in the data most effectively. Already a simple averaging approach, as applied for Stacking models 1 and 2, leads to increased performance when compared to the level 0 models. Interestingly, reducing the set of level 0 generalizers, like for Stacking model 2, does not add more value. Nevertheless, it shall be noted that such restriction of input predictions could still decrease computational costs. Applying a more sophisticated level 1 combiner, like Gradient Boosting or Logistic Regression, furthermore improves prediction. In particular, the Logistic Regression combiner is able to increase AUC performance.

6 Conclusion

This paper aimed to discuss and evaluate Stacking and Ensemble models in a financial application of credit risk assessment. Focus was set on introducing the concepts of Bagging, Boosting and Stacked Generalization as well as the corresponding Random Forest and Gradient Boosting models. In depth, it was explained how Bagging and Boosting aim at increasing performance by decreasing generalization variance of base learners and by decreasing generalization bias of base learners, respectively. Moreover, the Stacked Generalization model was outlined, that seeks for the optimal way to combine the specific strengths of level 0 generalizers in order to increase predictive performance even more.

A broad set of different Stacking and Ensemble models as well as standard machine learning models was applied and evaluated to a classification problem of credit risk assessment. Thereby, the Random Forest model and Gradient Boosting model outperformed standard machine learning models on behalf of the calculated metrics. Furthermore, all four Stacked Generalization approaches were able to establish a better performance on AUC, Logarithmic Loss and Brier Score than their level 0 generalizers. With regards to Accuracy metric, they performed worse, suggesting that the metric-implied cut-off threshold of 0.5 being suboptimal. Even a simple combiner algorithm like averaging could increase performance of the Stacking model, while the Logistic Regression combiner performed best. Restricting the subset of input predictions seems to be ineffective with regards to performance issues. To conclude, Stacking and Ensemble models could show their comparative strengths in this study. The results strongly confirm their value for prediction issues in credit risk assessment.

While this study focused on predicting binary outcomes, further research could evaluate the performance of Stacking and Ensemble models for regression problems in a similar context. A shortfall of Stacking and Ensemble models is their need of much computational resources caused by their complexity. In the context of *scalability*, current research tests ideas that restrict computational costs, e.g. by adaptive parallelization (Li et al., 2014) or by improving intra-model algorithms like stochastic gradient descent (Bottou, 2012). A further problem of such models is the absence of proven statistical properties like unbiasedness, consistency or asymptotic theory for construction of confidence intervals. In order to use machine learning models for research purposes, such properties are however necessary. Only recently, the field of *machine learning in economics* emerged, where the development of such properties is aimed at (see Athey (2017), Wager and Athey (in press)). Developments in this field of research will also enable better application of machine learning models to financial research problems.

A Appendix - Summary Tables

	Mean	Std. Dev.	Median	Minimum	Maximum
Duration	20.90	12.06	18.00	4.00	72.00
Amount	3271.26	2822.74	2319.50	250.00	18424.00
Installment Rate	2.97	1.12	3.00	1.00	4.00
Residence Duration	2.85	1.10	3.00	1.00	4.00
Age	35.55	11.38	33.00	19.00	75.00
Number of Credits	1.41	0.58	1.00	1.00	4.00
Number of Liable People	1.16	0.36	1.00	1.00	2.00

Table 2: Summary statistics for numerical features in the German Credit Dataset.

Feature	Category	Count	Fraction
Customer Classification (Outcome Feature)	good	700	70%
	bad	300	30%
Account Status	x < 0 DM (D-Mark)	274	27.4%
	0 DM < x < 200 DM	269	26.9%
	x >= 200 DM	63	6.3%
	no account	394	39.4%
Credit History	no credits taken/all paid back duly	40	4%
	all credits at this bank paid back duly	49	4.9%
	existing credits paid back duly till now	530	53%
	delay in paying off in the past	88	8.8%
	critical account	293	29.3%
Purpose	car (new)	234	23.4%
	car(used	103	10.3%
	furniture/equipment	12	1.2%
	radio/television	181	18.1%
	domestic appliances	280	28%
	repairs	12	1.2%
	education	22	2.2%
	vacation	50	5%
	retraining	9	0.9%
	business	97	9.7%
Savings	x < 100 DM	603	60.3%
	100 <= x < 500 DM	103	10.3%
	500 <= x < 1000 DM	63	6.3%
	x >= 1000 DM	48	4.8%
	unknown/no savings	183	18.3%

(continued on next page)

(continued)

Feature	Category	Count	Fraction
Employment Duration	unemployed	62	6.2%
	x < 1 year	172	17.2%
	1 ≤ x < 4 years	339	33.9%
	4 ≤ x < 7 years	174	17.4%
	x ≥ 7 years	253	25.3%
Status and Sex	male: divorced/separated	50	5%
	female: divorced/separated/married	310	31%
	male: single	548	54.8%
	male: married/widowed	92	9.2%
Other Debtors	none	907	90.7%
	co-applicant	41	4.1%
	guarantor	52	5.2%
Property	real estate	282	28.2%
	savings agreement/life insurance	232	23.2%
	car or other	332	33.2%
	unknown/no property	154	15.4%
Other Installment Plans	bank	139	13.9%
	stores	47	4.7%
	none	814	81.4%
Housing	rent	179	17.9%
	own	713	71.3%
	for free	108	10.8%
Job	unemployed/ unskilled - non-resident	22	2.2%
	unskilled - resident	200	20%
	skilled employee / official	630	63%
	management/self-employed/officer	148	14.8%
Telephone	none	596	59.6%
	yes	404	40.4%
Foreign Worker	yes	963	96.3%
	no	37	3.7%

Table 3: Summary statistics for categorical features in the German Credit Dataset.

B Appendix - Code



1: Main file

```

1 #####
2 # Numerical Introductory Course 2018
3 # Topic: Stacking and Ensemble Modelling
4 # Supervisor: Prof. Dr. Brenda López Cabrera
5 # Student: Frederik Schreck
6 #####
7
8
9 #-----
10 # Prepare environment
11 #-----
12
13 # Install packages and set working directory to path of main.R file
14 library("rstudioapi")
15 setwd(dirname(rstudioapi::getActiveDocumentContext())$path))
16
17 library("plyr")
18 library("mlr")
19 library("parallelMap")
20 library("parallel")
21 library("ggplot2")
22 library("corrplot")
23 library("glmnet")
24 library("xtable")
25 library("caret")
26 library("caretEnsemble")
27 library("gbm")
28 library("ROCR")
29
30 rm(list = ls(all = TRUE))
31 graphics.off()
32
33
34 #-----
35 # Loading dataset: German Credit Data
36 #-----
37
38 creditdata      <- read.delim("dataset/german_credit_data.txt",
39                               header = FALSE, sep = " ")
40
41 # Renaming of feature
42 colnames(creditdata) <- c("account_status", "duration",
43                            "credit_history", "purpose", "amount",
44                            "savings", "employment_duration",
45                            "installment_rate", "status_sex",
46                            "other_debtors", "residence_duration",
47                            "property", "age", "installment_plans",
48                            "housing", "number_credits", "job",
49                            "liable_people", "telephone",
50                            "foreign_worker", "customer")

```

```

51
52
53 #-----
54 # Feature Engineering
55 #-----
56
57 # Formatting and labelling
58 creditdata$duration      <- as.numeric(creditdata$duration)
59 creditdata$amount        <- as.numeric(creditdata$amount)
60 creditdata$installment_rate <- as.numeric(creditdata$installment_rate)
61 creditdata$residence_duration <- as.numeric(creditdata$residence_duration)
62 creditdata$age           <- as.numeric(creditdata$age)
63 creditdata$number_credits <- as.numeric(creditdata$number_credits)
64 creditdata$duration      <- as.numeric(creditdata$duration)
65 creditdata$liable_people <- as.numeric(creditdata$liable_people)
66 creditdata$customer      <- factor(creditdata$customer,
67                                   levels = c(1, 2),
68                                   labels = c("good", "bad"))
69
70 # No problem of missing values
71 apply(creditdata, 2, function(x) sum(is.na(x)))
72
73 # Create summary tables for numeric and for categorical features
74 numeric_vars <- lapply(Filter(is.numeric, creditdata),
75                         function(x) rbind(mean = mean(x),
76                                           sd = sd(x),
77                                           median = median(x),
78                                           minimum = min(x),
79                                           maximum = max(x)))
80 print(xtable(t(data.frame(numeric_vars))),
81       file="tables/summary_numeric.txt")
82
83
84 cat_vars <- ldply(Filter(is.factor, creditdata),
85                  function(x) t(rbind(names(table(x)),
86                                     table(x),
87                                     paste0(prop.table(table(x))*100,"%"))))
88 colnames(cat_vars) <- c("feature", "category", "Count", "Fraction")
89 print(xtable(data.frame(cat_vars)),
90       file="tables/summary_categorical.txt")
91
92 # Standardize numeric features for models performance
93 creditdata$duration      <- scale(creditdata$duration)
94 creditdata$amount        <- scale(creditdata$amount)
95 creditdata$installment_rate <- scale(creditdata$installment_rate)
96 creditdata$residence_duration <- scale(creditdata$residence_duration)
97 creditdata$age           <- scale(creditdata$age)
98 creditdata$number_credits <- scale(creditdata$number_credits)
99 creditdata$duration      <- scale(creditdata$duration)
100 creditdata$liable_people  <- scale(creditdata$liable_people)
101
102
103 #-----
104 # Data partitioning into training and testing data
105 #-----
106
107 set.seed(2601)

```

```

108 idx <- sample(x = NROW(creditdata),
109             size = floor(0.75*NROW(creditdata)),
110             replace = FALSE)
111 train <- creditdata[idx,]
112 test <- creditdata[-idx,]
113
114
115 #-----
116 # Feature Selection
117 #-----
118
119 'For each model, a wrapper approach is run on the training dataset
120 in order to find the best subset of features. The optimal set of
121 features is then used in the subsequent model building part.
122
123 The wrappers as well as the model tuning are evaluated on the AUC
124 measure. Tuning of the hyperparameters in the wrapper is done by
125 a grid-approach with 3-fold cross-validation process.
126
127 For reproduction of the results, the output of the wrapper
128 files can be accessed directly.'
129
130 # Generate empty lists to store the models and their results in
131 vars <- list() # Selected set of features used for each model
132 model_lib <- list() # Model library
133 yhat <- list() # Predictions on test dataset
134
135
136 ##### Gradient Boosting wrapper #####
137 source("gb_wrapper.R")
138
139 # For replication purposes: gb_wrapper results
140 train_wrapper <- mlr::createDummyFeatures(train, target = "customer")
141 vars$gb <- c("customer", "account_status.A11",
142             "account_status.A14", "credit_history.A30",
143             "credit_history.A34", "purpose.A49", "savings.A63",
144             "savings.A65", "employment_duration.A75",
145             "other_debtors.A103", "property.A123",
146             "installment_plans.A143", "job.A174",
147             "telephone.A192")
148
149
150 ##### Random forest wrapper #####
151 source("rf_wrapper.R")
152
153 # For replication purposes: rf_wrapper results
154 vars$rf <- c("customer", "account_status", "duration",
155             "credit_history", "purpose", "savings",
156             "employment_duration", "other_debtors",
157             "residence_duration", "age", "installment_plans",
158             "housing", "job", "liable_people", "foreign_worker")
159
160
161 ##### Stacked Generalization model #####
162 'For the Stacked Generalization model, firstly the training data is
163 split into five disjoint sets. Secondly, the Random Forest model and
164 the Gradient Boosting model are rebuilt on that subset. For each

```

```

165 model, a new wrapper approach is applied to find the optimal subset
166 of features.
167
168 Additionally, a Decision Tree, a Logistic Regression and a
169 Neural Network are built. Similar to the models before, a
170 model-specific wrapper approach is applied before the model
171 building process.
172
173 For reproduction of the results, the output of the wrapper
174 functions can again be accessed directly.
175 '
176
177 ##### Stacking: Decision tree wrapper #####
178 source("dt_wrapper.R")
179
180 # For replication purposes: dt_wrapper results
181 vars$dt      <- c("customer", "account_status", "credit_history",
182                 "savings", "number_credits")
183
184 ##### Stacking: Logit wrapper #####
185 source("logit_wrapper.R")
186
187 # For replication purposes: logit_wrapper results
188 vars$logit    <- c("customer", "duration", "credit_history",
189                 "purpose", "amount", "installment_rate",
190                 "status_sex", "residence_duration", "age",
191                 "installment_plans", "number_credits",
192                 "liable_people", "telephone", "foreign_worker")
193
194 ##### Stacking: Neural net wrapper #####
195 source("nnet_wrapper.R")
196
197 # For replication purposes: nnet_wrapper results
198 vars$nnet     <- c("customer", "account_status", "credit_history")
199
200 ##### Stacking: Gradient Boosting wrapper #####
201 source("gb_wrapper_stacking.R")
202
203 # For replication purposes: gb_wrapper results
204 vars$gb2      <- c("customer", "account_status.A11",
205                 "account_status.A14", "credit_history.A34",
206                 "purpose.A41", "savings.A65",
207                 "employment_duration.A74", "status_sex.A94",
208                 "other_debtors.A103", "installment_plans.A143")
209
210 ##### Stacking: Random forest wrapper #####
211 source("rf_wrapper_stacking.R")
212
213 # For replication purposes: rf_wrapper results
214 vars$rf2      <- c("customer", "account_status", "duration",
215                 "credit_history", "purpose", "savings",
216                 "employment_duration", "other_debtors",
217                 "residence_duration", "age",
218                 "installment_plans", "housing", "job",
219                 "liable_people", "foreign_worker")
220
221

```

```

222 #-----
223 # Model Building
224 #-----
225
226 'In the following, the models are built, parameters are tuned and
227 predictions on the test dataset are made. Each model uses the
228 optimal subset of features from the corresponding wrapper approach.'
229
230 ##### Random forest tuning #####
231 source("rf_model.R")
232
233
234 ##### Gradient Boosting tuning #####
235 source("gb_model.R")
236
237
238 ##### Stacking: preparation #####
239
240 # Partitioning the dataset for the Stacking into five equally
241 # sized disjoint sets
242 set.seed(2610)
243 idx2      <- sample(rep(1:5,each = nrow(train)/5))
244 train_sets <- lapply(split(1:nrow(train), idx2),
245                     function(i) creditdata[i,])
246
247 ##### Stacking: Decision tree tuning #####
248 source("dt_model.R")
249
250
251 ##### Stacking: Logit model #####
252 # No tuning necessary
253 source("logit_model.R")
254
255
256 ##### Stacking: Neural net tuning #####
257 source("nnet_model.R")
258
259
260 ##### Stacking: Gradient Boosting tuning #####
261 source("gb_model_stacking.R")
262
263
264 ##### Stacking: Random forest tuning #####
265 source("rf_model_stacking.R")
266
267
268 ##### Stacked Generalization Models #####
269 source("st_model.R")
270
271
272 #-----
273 # Model evaluation
274 #-----
275
276 'After having built all the models, they can now be compared and
277 evaluated with regard to a variety of evaluation metrics.'
278

```

```

279 # Create empty lists to store the measure values in
280 auc      <- list() # Area under curve performance measure
281 acc      <- list() # Accuracy performance measure
282 kappa    <- list() # Kappa performance measure
283 logloss  <- list() # Logarithmic Loss measure
284 brier    <- list() # Brier score measure
285
286 # AUC
287 auc$rf    <- mlr::performance(yhat$rf,
288                               measures = mlr::auc); auc$rf
289 auc$gb    <- mlr::performance(yhat$gb,
290                               measures = mlr::auc); auc$gb
291 auc$dt    <- mlr::performance(yhat$dt_test,
292                               measures = mlr::auc); auc$dt
293 auc$logit <- mlr::performance(yhat$logit_test,
294                               measures = mlr::auc); auc$logit
295 auc$nnnet <- mlr::performance(yhat$nnnet_test,
296                               measures = mlr::auc); auc$nnnet
297 auc$gb2   <- mlr::performance(yhat$gb2_test,
298                               measures = mlr::auc); auc$gb2
299 auc$rf2   <- mlr::performance(yhat$rf2_test,
300                               measures = mlr::auc); auc$rf2
301 auc$st1   <- measureAUC(probabilities = yhat$st1$prob.good,
302                          truth = yhat$st1$truth, positive = "1",
303                          negative = "2"); auc$st1
304 auc$st2   <- measureAUC(probabilities = yhat$st2$prob.good,
305                          truth = yhat$st2$truth, positive = "1",
306                          negative = "2"); auc$st2
307 auc$st3   <- measureAUC(probabilities = yhat$st3[,3],
308                          truth = yhat$st3[,2], positive = "1",
309                          negative = "2"); auc$st3
310 auc$st4   <- measureAUC(probabilities = yhat$st4[,3],
311                          truth = yhat$st4[,2], positive = "1",
312                          negative = "2"); auc$st4
313
314 # Accuracy
315 acc$rf    <- mlr::performance(yhat$rf,
316                               measures = mlr::acc); acc$rf
317 acc$gb    <- mlr::performance(yhat$gb,
318                               measures = mlr::acc); acc$gb
319 acc$dt    <- mlr::performance(yhat$dt_test,
320                               measures = mlr::acc); acc$dt
321 acc$logit <- mlr::performance(yhat$logit_test,
322                               measures = mlr::acc); acc$logit
323 acc$nnnet <- mlr::performance(yhat$nnnet_test,
324                               measures = mlr::acc); acc$nnnet
325 acc$gb2   <- mlr::performance(yhat$gb2_test,
326                               measures = mlr::acc); acc$gb2
327 acc$rf2   <- mlr::performance(yhat$rf2_test,
328                               measures = mlr::acc); acc$rf2
329 acc$st1   <- measureACC(response = round(yhat$st1$prob.good),
330                          truth = yhat$st1$truth); acc$st1
331 acc$st2   <- measureACC(response = round(yhat$st2$prob.good),
332                          truth = yhat$st2$truth); acc$st2
333 acc$st3   <- measureACC(response = round(yhat$st3[,3]),
334                          truth = yhat$st3[,2]); acc$st3
335 acc$st4   <- measureACC(response = round(yhat$st4[,3]),

```



```

336         truth = yhat$st4[,2]); acc$st4
337
338 # Logarithmic Loss
339 logloss$rf      <- mlr::performance(yhat$rf,
340                                     measures = mlr::logloss)
341 logloss$rf
342 logloss$gb      <- mlr::performance(yhat$gb,
343                                     measures = mlr::logloss)
344 logloss$gb
345 logloss$dt      <- mlr::performance(yhat$dt_test,
346                                     measures = mlr::logloss)
347 logloss$dt
348 logloss$logit   <- mlr::performance(yhat$logit_test,
349                                     measures = mlr::logloss)
350 logloss$logit
351 logloss$nnet    <- mlr::performance(yhat$nnet_test,
352                                     measures = mlr::logloss)
353 logloss$nnet
354 logloss$gb2     <- mlr::performance(yhat$gb2_test,
355                                     measures = mlr::logloss)
356 logloss$gb2
357 logloss$rf2     <- mlr::performance(yhat$rf2_test,
358                                     measures = mlr::logloss)
359 logloss$rf2
360 log_loss=function(actual, predicted)
361 {
362     result=-1/length(actual)*
363         (sum((actual*log(predicted)+
364             (1-actual)*log(1-predicted))))
365     return(result)
366 }
367 logloss$st1     <- log_loss(yhat$st1$truth, yhat$st1$prob.good)
368 logloss$st1
369 logloss$st2     <- log_loss(yhat$st2$truth, yhat$st2$prob.good)
370 logloss$st2
371 logloss$st3     <- log_loss(yhat$st3[,2], yhat$st3[,3])
372 logloss$st3
373 logloss$st4     <- log_loss(yhat$st4[,2], yhat$st4[,3])
374 logloss$st4
375
376 # MSE/Brier score
377 brier$rf        <- mlr::performance(yhat$rf,
378                                     measures = mlr::brier); brier$rf
379 brier$gb        <- mlr::performance(yhat$gb,
380                                     measures = mlr::brier); brier$gb
381 brier$dt        <- mlr::performance(yhat$dt_test,
382                                     measures = mlr::brier); brier$dt
383 brier$logit     <- mlr::performance(yhat$logit_test,
384                                     measures = mlr::brier); brier$logit
385 brier$nnet      <- mlr::performance(yhat$nnet_test,
386                                     measures = mlr::brier); brier$nnet
387 brier$gb2       <- mlr::performance(yhat$gb2_test,
388                                     measures = mlr::brier); brier$gb2
389 brier$rf2       <- mlr::performance(yhat$rf2_test,
390                                     measures = mlr::brier); brier$rf2
391 brier$st1       <- measureBrier(probabilities = yhat$st1$prob.good,
392                                truth = yhat$st1$truth,

```

```

393         positive = 1, negative = 2); brier$st1
394 brier$st2     <- measureBrier(probabilities = yhat$st2$prob.good,
395                               truth = yhat$st2$truth, positive = 1,
396                               negative = 2); brier$st2
397 brier$st3     <- measureBrier(probabilities = yhat$st3[,3],
398                               truth = yhat$st3[,2], positive = 1,
399                               negative = 2); brier$st3
400 brier$st4     <- measureBrier(probabilities = yhat$st4[,3],
401                               truth = yhat$st4[,2], positive = 1,
402                               negative = 2); brier$st4
403
404 # Make evaluation table
405 eval_table     <- cbind(auc, acc, logloss, brier)
406 rownames(eval_table) <- c("Random Forest", "Gradient Boosting",
407                            "Decision Tree (level 0)",
408                            "Logit Regression (level 0)",
409                            "Neural Network (level 0)",
410                            "Random Forest (level 0)",
411                            "Gradient Boosting (level 0)",
412                            "Stacking Model 1", "Stacking Model 2",
413                            "Stacking Model 3", "Stacking Model 4")
414 colnames(eval_table) <- c("AUC", "Accuracy",
415                            "Logarithmic Loss", "Brier Score")
416
417
418 print(xtable(eval_table), file="tables/evaltable.txt")
419
420 # Generate plot with AUC curves for all models
421 plot(performance(prediction(yhat$rf$data$prob.good,
422                             labels = matrix(test$customer,
423                                                 nrow = NROW(test$customer),
424                                                 ncol = 1))
425       , "tpr", "fpr"),
426       colorize = FALSE, col = "gray0", type = "l",
427       lty = 1, lwd = 1.2,
428       xlab="1 - Specificity", ylab="Sensitivity")
429
430 plot(performance(prediction(yhat$gb$data$prob.good,
431                             labels = matrix(test$customer,
432                                                 nrow = NROW(test$customer),
433                                                 ncol = 1))
434       , "tpr", "fpr"),
435       colorize = FALSE, add = TRUE, col = "gray40",
436       type = "l", lty = 1, lwd = 1.2)
437
438 plot(performance(prediction(yhat$dt_test$data$prob.good,
439                             labels = matrix(test$customer,
440                                                 nrow = NROW(test$customer),
441                                                 ncol = 1))
442       , "tpr", "fpr"),
443       colorize = FALSE, add = TRUE, col = "gray0",
444       type = "l", lty = 1, lwd = 1.2)
445
446 plot(performance(prediction(yhat$logit_test$data$prob.good,
447                             labels = matrix(test$customer,
448                                                 nrow = NROW(test$customer),
449                                                 ncol = 1))

```

```

450         , "tpr", "fpr"),
451     colorize = FALSE, add = TRUE, col = "gray25",
452     type = "l", lty = 2, lwd = 1.2)
453
454 plot(performance(prediction(yhat$nnet_test$data$prob.good,
455                             labels = matrix(test$customer,
456                                             nrow = NROW(test$customer),
457                                             ncol = 1))
458     , "tpr", "fpr"),
459     colorize = FALSE, add = TRUE, col = "gray50",
460     type = "l", lty = 2, lwd = 1.2)
461
462 plot(performance(prediction(yhat$rfr2_test$data$prob.good,
463                             labels = matrix(test$customer,
464                                             nrow = NROW(test$customer),
465                                             ncol = 1))
466     , "tpr", "fpr"),
467     colorize = FALSE, add = TRUE, col = "gray60",
468     type = "l", lty = 2, lwd = 1.2)
469
470 plot(performance(prediction(yhat$gb2_test$data$prob.good,
471                             labels = matrix(test$customer,
472                                             nrow = NROW(test$customer),
473                                             ncol = 1))
474     , "tpr", "fpr"),
475     colorize = FALSE, add = TRUE, col = "gray70",
476     type = "l", lty = 2, lwd = 1.2)
477
478 plot(performance(prediction(yhat$st1$prob.good,
479                             labels = matrix(test$customer,
480                                             nrow = NROW(test$customer),
481                                             ncol = 1))
482     , "tpr", "fpr"),
483     colorize = FALSE, add = TRUE, col = "gray0",
484     type = "l", lty = 3, lwd = 1.2)
485
486 plot(performance(prediction(yhat$st2$prob.good,
487                             labels = matrix(test$customer,
488                                             nrow = NROW(test$customer),
489                                             ncol = 1))
490     , "tpr", "fpr"),
491     colorize = FALSE, add = TRUE, col = "gray25",
492     type = "l", lty = 3, lwd = 1.2)
493
494 plot(performance(prediction(yhat$st3[,3],
495                             labels = matrix(test$customer,
496                                             nrow = NROW(test$customer),
497                                             ncol = 1))
498     , "tpr", "fpr"),
499     colorize = FALSE, add = TRUE, col = "gray50",
500     type = "l", lty = 3, lwd = 1.2)
501
502 plot(performance(prediction(yhat$st4[,3],
503                             labels = matrix(test$customer,
504                                             nrow = NROW(test$customer),
505                                             ncol = 1))
506     , "tpr", "fpr"),

```

```

507     colorize = FALSE, add = TRUE, col = "gray75",
508     type = "l", lty = 3, lwd = 1.2)
509
510 lines(x = c(0,1), y = c(0,1))
511
512 legend(xy.coords(0.6,0.57), legend = c("Random Forest",
513     "Gradient Boosting",
514     "Decision Tree (level 0)",
515     "Logit Regression (level 0)",
516     "Neural Network (level 0)",
517     "Random Forest (level 0)",
518     "Gradient Boosting (level 0)",
519     "Stacking Model 1",
520     "Stacking Model 2",
521     "Stacking Model 3",
522     "Stacking Model 4"),
523     lty= c(1,1,2,2,2,2,2,3,3,3,3), cex=0.75, col=c("gray0", "gray40",
524     "gray0", "gray25",
525     "gray50", "gray60",
526     "gray70", "gray0",
527     "gray25", "gray50",
528     "gray75"),
529     box.lty=0)

```



2: Gradient Boosting Model Feature Selection

```

1 ##### Variable selection for the Gradient Boosting model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on the AUC measure.
6
7 set.seed(2610)
8
9 # One-hot-encoding of categorical features
10 train_wrapper      <- mlr::createDummyFeatures(train,
11     target = "customer")
12
13 gb_task_wrapper    <- makeClassifTask(data = train_wrapper,
14     target = "customer",
15     positive = "good")
16
17 gb_learner_wrapper<- makeLearner("classif.xgboost",
18     predict.type = "prob")
19
20 gb_ctrl_wrapper    <- makeFeatSelControlSequential(method = "sfs",
21     alpha = 0.00001)
22
23 gb_rdesc_wrapper   <- makeResampleDesc("CV", iters = 3)
24
25 gb_sfeats          <- selectFeatures(learner = gb_learner_wrapper,
26     task = gb_task_wrapper,
27     resampling = gb_rdesc_wrapper,
28     control = gb_ctrl_wrapper,

```

```

29         show.info = TRUE,
30         measures = mlr::auc)
31
32 # Performance score for each combination of features
33 analyzeFeatSelResult(gb_sfeats)
34
35 # Next, I define the dataset to use for the gradient
36 # boosting model. This dataset is then used in the model
37 # building file "gb_model.R"
38 vars$gb      <- c("customer", gb_sfeats$x)

```



3: Random Forest Model Feature Selection

```

1 ##### Feature selection for the Random Forest model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is established in order
4 # to find the best subset of features. Each model is built with crossvalidation on
5 # the AUC measure.
6
7 set.seed(2610)
8
9 rf_task_wrapper      <- makeClassifTask(data = train,
10                                         target = "customer",
11                                         positive = "good")
12
13 rf_learner_wrapper   <- makeLearner("classif.randomForest",
14                                     predict.type = "prob",
15                                     "ntree" = 300)
16
17 rf_ctrl_wrapper      <- makeFeatSelControlSequential(method = "sfs",
18                                                         alpha = 0.00001)
19
20 rf_rdesc_wrapper     <- makeResampleDesc("CV", iters = 3)
21
22 rf_sfeats            <- selectFeatures(learner = rf_learner_wrapper,
23                                       task = rf_task_wrapper,
24                                       resampling = rf_rdesc_wrapper,
25                                       control = rf_ctrl_wrapper,
26                                       show.info = TRUE,
27                                       measures = mlr::auc)
28
29 # Performance score for each combination
30 analyzeFeatSelResult(rf_sfeats)
31
32 # Next, I define the dataset to use for the gradient boosting model.
33 # This dataset is then used in the model building file "rf_model.R"
34 vars$rf <- c("customer", rf_sfeats$x)

```



4: Stacking: Decision Tree (level 0) Feature Selection

```

1 ##### Feature selection for the Decision Tree model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on the AUC measure.
6
7 set.seed(2610)
8
9 dt_task_wrapper      <- makeClassifTask(data = train, target = "customer",
10                                         positive = "good")
11
12 dt_learner_wrapper   <- makeLearner("classif.rpart",
13                                     predict.type = "prob")
14
15 # We set the minimum required difference to 0.0001 Euros in expected loss
16 # per observation
17 dt_ctrl_wrapper      <- makeFeatSelControlSequential(method = "sfs",
18                                                         alpha = 0.00001)
19
20 dt_rdesc_wrapper     <- makeResampleDesc("CV", iters = 3)
21
22 dt_sfeats            <- selectFeatures(learner = dt_learner_wrapper,
23                                     task = dt_task_wrapper,
24                                     resampling = dt_rdesc_wrapper,
25                                     control = dt_ctrl_wrapper,
26                                     show.info = TRUE,
27                                     measures = mlr::auc)
28
29 # Performance score for each combination of features
30 analyzeFeatSelResult(dt_sfeats)
31
32 # Next, I store the optimal set of features to later use it
33 # in the model building part file "dt_model.R"
34 vars$dt <- c("customer", dt_sfeats$x)

```



5: Logistic Regression (level 0) Feature Selection

```

1 ##### Feature selection for the Logit model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on
6 # the AUC measure.
7
8 set.seed(2610)
9
10 logit_task_wrapper    <- makeClassifTask(data = train[, -1], target = "customer",
11                                           positive = "good")
12
13 logit_learner_wrapper <- makeLearner("classif.logreg",

```

```

14         predict.type = "prob")
15
16 logit_ctrl_wrapper    <- makeFeatSelControlSequential(method = "sfs",
17                                                         alpha = 0.00001)
18
19 logit_rdesc_wrapper   <- makeResampleDesc("CV", iters = 3)
20
21 logit_sfeats          <- selectFeatures(learner = logit_learner_wrapper,
22                                         task = logit_task_wrapper,
23                                         resampling = logit_rdesc_wrapper,
24                                         control = logit_ctrl_wrapper,
25                                         show.info = TRUE,
26                                         measures = mlr::auc)
27
28 # Performance score for each combination
29 analyzeFeatSelResult(logit_sfeats)
30
31 # Next, I store the optimal set of features to later use it
32 # in the model building part file "logit_model.R"
33 vars$logit <- c("customer", logit_sfeats$x)

```



6: Stacking: Neural Network (level 0) Feature Selection

```

1 ##### Feature selection for the Neural Network. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on the AUC measure.
6
7 set.seed(2610)
8
9 nnet_task_wrapper     <- makeClassifTask(data = train,
10                                         target = "customer",
11                                         positive = "good")
12
13 nnet_learner_wrapper  <- makeLearner("classif.nnet",
14                                     predict.type = "prob",
15                                     "trace" = FALSE)
16
17 nnet_ctrl_wrapper     <- makeFeatSelControlSequential(method = "sfs",
18                                                         alpha = 0.00001)
19
20 nnet_rdesc_wrapper    <- makeResampleDesc("CV", iters = 3)
21
22 nnet_sfeats           <- selectFeatures(learner = nnet_learner_wrapper,
23                                         task = nnet_task_wrapper,
24                                         resampling = nnet_rdesc_wrapper,
25                                         control = nnet_ctrl_wrapper,
26                                         show.info = TRUE,
27                                         measures = mlr::auc)
28
29 # Performance score for each combination
30 analyzeFeatSelResult(nnet_sfeats)
31

```

```

32 # Next, I store the optimal set of features to later use it
33 # in the model building part file "nnet_model.R"
34 vars$nnet <- c("customer", nnet_sfeats$x)

```



7: Gradient Boosting (level 0) Feature Selection

```

1 ##### Feature selection for the Gradient Boosting model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on the AUC measure.
6
7 # One-hot-encoding of categorical features
8 train_wrapper2 <- mlr::createDummyFeatures(train, target = "customer")
9
10 gb_task_wrapper <- makeClassifTask(data = train_wrapper2,
11                                   target = "customer",
12                                   positive = "good")
13
14 gb_learner_wrapper <- makeLearner("classif.xgboost",
15                                  predict.type = "prob")
16
17 gb_ctrl_wrapper <- makeFeatSelControlSequential(method = "sfs",
18                                                  alpha = 0.00001)
19
20 gb_rdesc_wrapper <- makeResampleDesc("CV", iters = 3)
21
22 gb_sfeats <- selectFeatures(learner = gb_learner_wrapper,
23                             task = gb_task_wrapper,
24                             resampling = gb_rdesc_wrapper,
25                             control = gb_ctrl_wrapper,
26                             show.info = TRUE,
27                             measures = mlr::auc)
28
29 # Performance score for each combination of features
30 analyzeFeatSelResult(gb_sfeats)
31
32 # Next, I store the optimal set of features to later use it
33 # in the model building part file "gb_model_stacking.R"
34 vars$gb2 <- c("customer", gb_sfeats$x)

```




8: Stacking: Random Forest (level 0) Feature Selection

```

1 ##### Feature selection for the Random Forest model. A wrapper approach #####
2
3 # A model building approach with sequential forward selection is
4 # established in order to find the best subset of features.
5 # Each model is built with crossvalidation on the AUC measure.
6
7 set.seed(2610)
8
9 rf_task_wrapper      <- makeClassifTask(data = train,
10                                     target = "customer",
11                                     positive = "good")
12
13 rf_learner_wrapper   <- makeLearner("classif.randomForest",
14                                     predict.type = "prob",
15                                     "ntree" = 300)
16
17 rf_ctrl_wrapper      <- makeFeatSelControlSequential(method = "sfs",
18                                     alpha = 0.00001)
19
20 rf_rdesc_wrapper     <- makeResampleDesc("CV", iters = 3)
21
22 rf_sfeats            <- selectFeatures(learner = rf_learner_wrapper,
23                                     task = rf_task_wrapper,
24                                     resampling = rf_rdesc_wrapper,
25                                     control = rf_ctrl_wrapper,
26                                     show.info = TRUE,
27                                     measures = mlr::auc)
28
29 # Performance score for each combination
30 analyzeFeatSelResult(rf_sfeats)
31
32 # Next, I store the optimal set of features to later use it
33 # in the model building part file "rf_model_stacking.R"
34 vars$rf2 <- c("customer", rf_sfeats$x)

```



9: Random Forest Model Building

```

1 ##### Random forest model #####
2
3 set.seed(2610)
4
5 # Define task
6 rf_task      <- makeClassifTask(data = train[, c(vars$rf)],
7                                     target = "customer", positive = "good")
8
9 # Define learner: decision tree
10 rf_learner <- makeLearner("classif.randomForest",
11                                     predict.type = "prob",
12                                     par.vals = list("replace" = TRUE,
13                                     "importance" = FALSE,

```

```

14         "ntree" = 800))
15
16 # Tuning the hyperparameters of the random forest
17 rf_parms <- makeParamSet(
18   # Number of features selected at each node
19   makeIntegerParam("mtry", lower = 2, upper = 12),
20   # Bootstrap sample size
21   makeDiscreteParam("sampsize", values = c(30, 50, 70, 100, 130)),
22   # Node size
23   makeIntegerParam("nodesize", lower = 2, upper = 12)
24 )
25 rf_tunecontrol <- makeTuneControlGrid(resolution = 7, tune.threshold = FALSE)
26
27 # Cross validation
28 rf_rdesc <- makeResampleDesc(method = "CV", iters = 3, stratify = TRUE)
29
30 # Tuning with parallel computing
31 no_cores <- detectCores() - 1
32
33 parallelStartSocket(no_cores, level = "mlr.tuneParams")
34 system.time(
35   rf_tuning <- tuneParams(rf_learner, task = rf_task,
36                           resampling = rf_rdesc,
37                           par.set = rf_parms,
38                           control = rf_tunecontrol,
39                           measures = mlr::auc)
40 )
41 parallelStop()
42
43 # Results for the different choices of hyperparameters
44 rf_tuning_results <- generateHyperParsEffectData(rf_tuning, partial.dep = TRUE)
45 rf_tuning_results$data
46
47 # Detailed investigation
48 tapply(rf_tuning_results$data$auc.test.mean,
49        INDEX = c(rf_tuning_results$data$mtry), mean)
50 tapply(rf_tuning_results$data$auc.test.mean,
51        INDEX = c(rf_tuning_results$data$sampsize), mean)
52 tapply(rf_tuning_results$data$auc.test.mean,
53        INDEX = c(rf_tuning_results$data$nodesize), mean)
54
55 # Choose the optimal hyperparameters and update the learner
56 rf_tuned <- setHyperPars(rf_learner, par.vals = rf_tuning$x)
57 rf_tuned
58
59 # Now we train the model on the full training data
60 model_lib$rf <- mlr::train(rf_tuned, task = rf_task)
61
62 # Prediction on current test data
63 yhat$rf <- predict(model_lib$rf, newdata = test)

```



10: Gradient Boosting Model Building

```

1 ##### Gradient Boosting model (with mlr package) #####
2
3 set.seed(2610)
4
5 # Define task
6 gb_task      <- makeClassifTask(data = train_wrapper[, c(vars$gb)],
7                                target = "customer", positive = "good")
8
9 # Define learner: gradient boosting model consisting of trees
10 gb_learner  <- makeLearner("classif.xgboost",
11                            predict.type = "prob",
12                            par.vals = list("booster" = "gbtree",
13                                             "silent" = 0))
14
15 # Tuning the hyperparameter of the model
16 gb_parms    <- makeParamSet(
17   # Learning rate
18   makeDiscreteParam("eta", values = c(0.15, 0.25, 0.3, 0.6)),
19   # Maximum depth of a tree
20   makeIntegerParam("max_depth", lower = 3, upper = 10),
21   # Minimum number of observations to have in a node
22   makeIntegerParam("min_child_weight", lower = 1, upper = 4),
23   # Number of iterations through data
24   makeIntegerParam("nrounds", lower = 4, upper = 12),
25   # L2 regularization on weights
26   makeDiscreteParam("lambda", values = c(0.05, 0.1, 0.2)),
27   # Minimum loss reduction
28   makeDiscreteParam("gamma", values = c(0.5, 0.9, 1.5)),
29   # Subsample size
30   makeDiscreteParam("subsample", values = c(1))
31 )
32
33 # Define how dense the parameters are selected from the defined ranges
34 gb_tunecontrol <- makeTuneControlGrid(resolution = 3,
35                                     tune.threshold = FALSE)
36
37 # Sampling strategy: cross validation
38 gb_rdesc      <- makeResampleDesc(method = "CV",
39                                  iters = 3,
40                                  stratify = TRUE)
41
42 # Tuning
43 no_cores      <- detectCores() - 1 # Detect number of cores
44
45 parallelStartSocket(no_cores, level = "mlr.tuneParams")
46 system.time(
47   gb_tuning    <- tuneParams(gb_learner,
48                             task = gb_task,
49                             resampling = gb_rdesc,
50                             par.set = gb_parms,
51                             control = gb_tunecontrol,
52                             measures = mlr::auc)
53 )

```

```

54 parallelStop()
55
56 # Results for the different choices of hyperparameters
57 gb_tuning_results <- generateHyperParsEffectData(gb_tuning,
58                                           partial.dep = TRUE)
59 gb_tuning_results$data
60
61 # Detailed investigation
62 tapply(gb_tuning_results$data$auc.test.mean,
63       INDEX = c(gb_tuning_results$data$eta), mean)
64 tapply(gb_tuning_results$data$auc.test.mean,
65       INDEX = c(gb_tuning_results$data$min_child_weight), mean)
66 tapply(gb_tuning_results$data$auc.test.mean,
67       INDEX = c(gb_tuning_results$data$nrounds), mean)
68 tapply(gb_tuning_results$data$auc.test.mean,
69       INDEX = c(gb_tuning_results$data$lambda), mean)
70 tapply(gb_tuning_results$data$auc.test.mean,
71       INDEX = c(gb_tuning_results$data$gamma), mean)
72 tapply(gb_tuning_results$data$auc.test.mean,
73       INDEX = c(gb_tuning_results$data$subsampling), mean)
74 tapply(gb_tuning_results$data$auc.test.mean,
75       INDEX = c(gb_tuning_results$data$max_depth), mean)
76
77 # Choose the optimal hyperparameters and update the learner
78 gb_tuned <- setHyperPars(gb_learner, par.vals = gb_tuning$x)
79 gb_tuned
80
81 # Now we train the model on the full training data
82 model_lib$gb <- mlr::train(gb_tuned, task = gb_task)
83
84 # Prediction on current (one hot encoded) test dataset
85 test_onehot <- mlr::createDummyFeatures(test, target = "customer")
86 yhat$gb <- predict(model_lib$gb, newdata = test_onehot[, c(vars$gb)])

```



11: Stacking: Decision Tree (level 0) Model Building

```

1 ##### Decision tree model #####
2
3 set.seed(2610)
4 pred <- list()
5
6 # Define dataset
7 for (i in 1:5) {
8   test_st <- train_sets[[i]]
9   train_st <- train[-as.numeric(rownames(train_sets[[i]])), ]
10
11   # Define task
12   dt_task <- makeClassifTask(data = train_st[, c(vars$dt)],
13                             target = "customer",
14                             positive = "good")
15
16   # Define learner: decision tree
17   dt_learner <- makeLearner("classif.rpart",
18                             predict.type = "prob")

```

```

19
20 # Tuning the hyperparameter of the tree
21 dt_parms <- makeParamSet(
22   # Complexity parameter
23   makeNumericParam("cp", lower = 0.00005, upper = 0.001),
24   # Minimum number of observation in a node for a split
25   makeDiscreteParam("minsplit", values = c(5, 7, 10, 12, 15, 17,
26     20, 23, 25, 27, 30)),
27   # Minimum number of observation to keep in terminal nodes
28   makeDiscreteParam("minbucket", values = c(5, 8, 10, 12,
29     15, 18, 21, 25))
30 )
31
32 # Grid density
33 dt_tunecontrol <- makeTuneControlGrid(resolution = 10,
34   tune.threshold = FALSE)
35
36 # Sampling strategy: cross validation
37 dt_rdesc <- makeResampleDesc(method = "CV",
38   iters = 3,
39   stratify = TRUE)
40
41 # Tuning using parallelization
42 no_cores <- detectCores() - 1 # Detect number of cores
43
44 parallelStartSocket(no_cores, level = "mlr.tuneParams")
45 system.time(
46   dt_tuning <- tuneParams(dt_learner,
47     task = dt_task,
48     resampling = dt_rdesc,
49     par.set = dt_parms,
50     control = dt_tunecontrol,
51     measures = mlr::auc)
52 )
53 parallelStop()
54
55 # Results for the different choices of hyperparameters
56 dt_tuning_results <- generateHyperParsEffectData(dt_tuning,
57   partial.dep = TRUE)
58 dt_tuning_results$data
59
60 # Detailed investigation
61 tapply(dt_tuning_results$data$auc.test.mean,
62   INDEX = c(dt_tuning_results$data$cp), mean)
63 tapply(dt_tuning_results$data$auc.test.mean,
64   INDEX = c(dt_tuning_results$data$minsplit), mean)
65 tapply(dt_tuning_results$data$auc.test.mean,
66   INDEX = c(dt_tuning_results$data$minbucket), mean)
67
68 # Choose the optimal hyperparameters and update the learner
69 dt_tuned <- setHyperPars(dt_learner, par.vals = dt_tuning$x)
70 dt_tuned
71
72 # Now the model is trained on the corresponding training data
73 model_lib$dt <- mlr::train(dt_tuned, task = dt_task)
74
75 # Prediction on current test_st data

```

```

76   pred[[i]]      <- predict(model_lib$dt, newdata = test_st)
77 }
78
79 # Combine subset predictions to obtain prediction on train dataset
80 yhat$dt_train    <- rbind(pred[[1]]$data, pred[[2]]$data, pred[[3]]$data,
81                           pred[[4]]$data, pred[[5]]$data)
82
83 # Prediction on test data
84 yhat$dt_test     <- predict(model_lib$dt, newdata = test)

```



12: Stacking: Logistic Regression (level 0) Model Building

```

1 ##### Logit model #####
2
3 set.seed(2610)
4 pred <- list()
5
6 # Define dataset
7 for (i in 1:5) {
8   test_st      <- train_sets[[i]]
9   train_st     <- train[-as.numeric(rownames(train_sets[[i]])), ]
10
11 # Define task
12 logit_task    <- makeClassifTask(data = train_st[, c(vars$logit)],
13                                 target = "customer",
14                                 positive = "good")
15
16 # Define learner: logistic regression
17 logit_learner <- makeLearner("classif.logreg",
18                             predict.type = "prob")
19
20 # No tuning of hyperparameters necessary for logit model
21
22 # Train the model on the full corresponding training data
23 model_lib$logit <- mlr::train(logit_learner, task = logit_task)
24
25 # Prediction on current test_st data
26 pred[[i]]     <- predict(model_lib$logit, newdata = test_st)
27 }
28
29 # Combine subset predictions to obtain full prediction on train data
30 yhat$logit_train <- rbind(pred[[1]]$data, pred[[2]]$data, pred[[3]]$data,
31                           pred[[4]]$data, pred[[5]]$data)
32
33 # Prediction on test data
34 yhat$logit_test  <- predict(model_lib$logit, newdata = test)

```



13: Stacking: Neural Network (level 0) Model Building

```

1 ##### Neural Network #####
2
3 set.seed(2610)
4 pred <- list()
5
6 # Define dataset
7 for (i in 1:5) {
8   test_st <- train_sets[[i]]
9   train_st <- train[-as.numeric(rownames(train_sets[[i]])), ]
10
11 # Define task
12 nnet_task <- makeClassifTask(data = train_st[, c(vars$nnet)],
13                               target = "customer",
14                               positive = "good")
15
16 # Define learner: neural net
17 nnet_learner <- makeLearner("classif.nnet",
18                               predict.type = "prob",
19                               par.vals = list("trace" = FALSE,
20                                                 "maxit" = 400,
21                                                 "MaxNWts" = 3500))
22
23 # Tuning the hyperparameters of the random forest
24 nnet_parms <- makeParamSet(
25   makeDiscreteParam("decay", values = c(0, 0.1, 0.2, 0.3,
26                                           0.4, 0.5, 0.6)),
27   makeDiscreteParam("size", values = c(2, 4, 5, 6, 7, 8,
28                                           9, 10, 11, 12))
29 )
30
31 nnet_tunecontrol <- makeTuneControlGrid(resolution = 8,
32                                         tune.threshold = FALSE)
33
34 # Sampling strategy: cross validation
35 nnet_rdesc <- makeResampleDesc(method = "CV",
36                                 iters = 3,
37                                 stratify = TRUE)
38
39 # Tuning with parallel computing
40 no_cores <- detectCores() - 1 # Detect number of cores
41
42 parallelStartSocket(no_cores, level = "mlr.tuneParams")
43 system.time(
44   nnet_tuning <- tuneParams(nnet_learner,
45                             task = nnet_task,
46                             resampling = nnet_rdesc,
47                             par.set = nnet_parms,
48                             control = nnet_tunecontrol,
49                             measures = mlr::auc)
50 )
51 parallelStop()
52
53 # Results for the different choices of hyperparameters

```

```

54 nnet_tuning_results <- generateHyperParsEffectData(nnet_tuning,
55                                                    partial.dep = TRUE)
56 nnet_tuning_results$data
57
58 # Detailed investigation
59 tapply(nnet_tuning_results$data$auc.test.mean,
60        INDEX = c(nnet_tuning_results$data$decay), mean)
61 tapply(nnet_tuning_results$data$auc.test.mean,
62        INDEX = c(nnet_tuning_results$data$size), mean)
63
64 # Choose the optimal hyperparameters and update the learner
65 nnet_tuned <- setHyperPars(nnet_learner, par.vals = nnet_tuning$x)
66 nnet_tuned
67
68 # Now we train the model on the full corresponding training data
69 model_lib$nnet <- mlr::train(nnet_tuned, task = nnet_task)
70
71 # Prediction on current test_st data
72 pred[[i]] <- predict(model_lib$nnet, newdata = test_st)
73 }
74
75 # Combine subset predictions to obtain prediction on train dataset
76 yhat$nnet_train <- rbind(pred[[1]]$data, pred[[2]]$data, pred[[3]]$data,
77                          pred[[4]]$data, pred[[5]]$data)
78
79 # Prediction on test data
80 yhat$nnet_test <- predict(model_lib$nnet, newdata = test)

```



14: Stacking: Gradient Boosting (level 0) Model Building

```

1 ##### Stacking: Gradient Boosting model (with mlr package) #####
2
3 set.seed(2610)
4 pred <- list()
5
6 # Define dataset
7 for (i in 1:5) {
8   test_st <- train_sets[[i]]
9   train_st <- train[-as.numeric(rownames(train_sets[[i]])), ]
10
11   test_st <- mlr::createDummyFeatures(test_st, target = "customer")
12   train_st <- mlr::createDummyFeatures(train_st, target = "customer")
13   # Define task
14   gb_task <- makeClassifTask(data = train_st[, c(vars$gb2)],
15                             target = "customer",
16                             positive = "good")
17
18   # Define learner: gradient boosting model consisting of trees
19   gb_learner <- makeLearner("classif.xgboost",
20                             predict.type = "prob",
21                             par.vals = list("booster" = "gbtree",
22                                              "silent" = 0))
23
24   # Tuning the hyperparameter of the model

```



```

25 gb_parms      <- makeParamSet(
26   # Learning rate
27   makeDiscreteParam("eta", values = c(0.35, 0.45, 0.5, 0.55, 0.6)),
28   # Maximum depth of a tree
29   makeIntegerParam("max_depth", lower = 3, upper = 10),
30   # Minimum number of obs. to have per node
31   makeIntegerParam("min_child_weight", lower = 2, upper = 4),
32   # Number of iterations through data
33   makeIntegerParam("nrounds", lower = 8, upper = 16),
34   # L2 regularization on weights
35   makeDiscreteParam("lambda", values = c(0.05, 0.1, 0.15, 0.2, 0.3)),
36   # Minimum loss reduction
37   makeDiscreteParam("gamma", values = c(0.3, 0.4, 0.5, 0.6)),
38   # Subsample size
39   makeDiscreteParam("subsample", values = c(0.9, 0.95, 1))
40 )
41
42 # Define how dense the parameters are selected from the defined ranges
43 gb_tunecontrol <- makeTuneControlGrid(resolution = 3,
44                                     tune.threshold = FALSE)
45
46 # Sampling strategy: cross validation
47 gb_rdesc      <- makeResampleDesc(method = "CV",
48                                   iters = 5,
49                                   stratify = TRUE)
50
51 # Tuning
52 no_cores      <- detectCores() - 1 # Detect number of cores
53
54 parallelStartSocket(no_cores, level = "mlr.tuneParams")
55 system.time(
56   gb_tuning    <- tuneParams(gb_learner,
57                             task = gb_task,
58                             resampling = gb_rdesc,
59                             par.set = gb_parms,
60                             control = gb_tunecontrol,
61                             measures = mlr::auc)
62 )
63 parallelStop()
64
65 # Results for the different choices of hyperparameters
66 gb_tuning_results <- generateHyperParsEffectData(gb_tuning,
67                                                  partial.dep = TRUE)
68 gb_tuning_results$data
69
70 # Detailed investigation
71 tapply(gb_tuning_results$data$auc.test.mean,
72        INDEX = c(gb_tuning_results$data$eta), mean)
73 tapply(gb_tuning_results$data$auc.test.mean,
74        INDEX = c(gb_tuning_results$data$min_child_weight), mean)
75 tapply(gb_tuning_results$data$auc.test.mean,
76        INDEX = c(gb_tuning_results$data$nrounds), mean)
77 tapply(gb_tuning_results$data$auc.test.mean,
78        INDEX = c(gb_tuning_results$data$lambda), mean)
79 tapply(gb_tuning_results$data$auc.test.mean,
80        INDEX = c(gb_tuning_results$data$gamma), mean)
81 tapply(gb_tuning_results$data$auc.test.mean,

```

```

82     INDEX = c(gb_tuning_results$data$subsample), mean)
83   tapply(gb_tuning_results$data$auc.test.mean,
84     INDEX = c(gb_tuning_results$data$max_depth), mean)
85
86   # Choose the optimal hyperparameters and update the learner
87   gb_tuned      <- setHyperPars(gb_learner, par.vals = gb_tuning$x)
88   gb_tuned
89
90   # Now we train the model on the full training data
91   model_lib$gb2 <- mlr::train(gb_tuned, task = gb_task)
92
93   # Prediction on current test_st data
94   pred[[i]]     <- predict(model_lib$gb2, newdata = test_st[, c(vars$gb2)])
95 }
96
97 # Combine subset predictions to obtain full prediction on train data
98 yhat$gb2_train <- rbind(pred[[1]]$data, pred[[2]]$data, pred[[3]]$data,
99   pred[[4]]$data, pred[[5]]$data)
100
101 # Performance measured on one-hot encoded test data
102 test_onehot <- mlr::createDummyFeatures(test, target = "customer")
103 yhat$gb2_test <- predict(model_lib$gb2, newdata = test_onehot[, c(vars$gb2)])

```



15: Stacking: Random Forest (level 0) Model Building

```

1 ##### Stacking: Random forest model #####
2
3 set.seed(2610)
4 pred <- list()
5
6 # Define dataset
7 for (i in 1:5) {
8   test_st      <- train_sets[[i]]
9   train_st     <- train[-as.numeric(rownames(train_sets[[i]])), ]
10
11 # Define task
12 rf_task       <- makeClassifTask(data = train_st[, c(vars$rf2)],
13   target = "customer",
14   positive = "good")
15
16 # Define learner: decision tree
17 rf_learner    <- makeLearner("classif.randomForest",
18   predict.type = "prob",
19   par.vals = list("replace" = TRUE,
20     "importance" = FALSE,
21     "ntree" = 800))
22
23 # Tuning the hyperparameters of the random forest
24 rf_parms      <- makeParamSet(
25   # Number of features selected at each node.
26   makeIntegerParam("mtry", lower = 2, upper = 12),
27   # Bootstrap sample size
28   makeDiscreteParam("sampsize", values = c(30, 50, 70, 100, 130)),
29   # Size of nodes

```

```

30     makeIntegerParam("nodesize", lower = 2, upper = 12)
31   )
32   # Grid density
33   rf_tunecontrol <- makeTuneControlGrid(resolution = 7,
34                                     tune.threshold = FALSE)
35
36   # Sampling strategy: cross validation
37   rf_rdesc      <- makeResampleDesc(method = "CV",
38                                   iters = 3,
39                                   stratify = TRUE)
40
41   # Tuning with parallel computing
42   no_cores      <- detectCores() - 1 # Detect number of cores
43
44   parallelStartSocket(no_cores, level = "mlr.tuneParams")
45   system.time(
46     rf_tuning    <- tuneParams(rf_learner, task = rf_task,
47                               resampling = rf_rdesc,
48                               par.set = rf_parms,
49                               control = rf_tunecontrol,
50                               measures = mlr::auc)
51   )
52   parallelStop()
53
54   # Results for the different choices of hyperparameters
55   rf_tuning_results <- generateHyperParsEffectData(rf_tuning,
56                                                  partial.dep = TRUE)
57   rf_tuning_results$data
58
59   # Detailed investigation
60   tapply(rf_tuning_results$data$auc.test.mean,
61         INDEX = c(rf_tuning_results$data$mtry), mean)
62   tapply(rf_tuning_results$data$auc.test.mean,
63         INDEX = c(rf_tuning_results$data$samplesize), mean)
64   tapply(rf_tuning_results$data$auc.test.mean,
65         INDEX = c(rf_tuning_results$data$nodesize), mean)
66
67   # Choose the optimal hyperparameters and update the learner
68   rf_tuned      <- setHyperPars(rf_learner, par.vals = rf_tuning$x)
69   rf_tuned
70
71   # Now we train the model on the full training data
72   model_lib$rf2 <- mlr::train(rf_tuned, task = rf_task)
73
74   # Prediction on current test_st data
75   pred[[i]]     <- predict(model_lib$rf2, newdata = test_st)
76 }
77
78 # Combine subset predictions to obtain prediction on full train data
79 yhat$rf2_train <- rbind(pred[[1]]$data, pred[[2]]$data, pred[[3]]$data,
80                        pred[[4]]$data, pred[[5]]$data)
81
82 # Performance measured on test data
83 yhat$rf2_test  <- predict(model_lib$rf2, newdata = test)

```



16: Stacking: Stacked Generalization Model Building

```

1 ##### Stacked Generalization models #####
2
3 # Generate new train and test dataset with base learners' predictions as features
4 data_stacking_train <- as.data.frame(matrix(data = c(train$customer,
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
##### Stacking Model 1: Averaging, all base learners #####
yhat$st1
##### Stacking Model 2: Averaging, best learners #####

```

```

52 yhat$st2          <- as.data.frame(matrix(data = c(data_stacking_test$truth,
53                                                    rowMeans(data_stacking_test[, -c(1,
54                                                    2, 3)])),
55                                                    ncol = 2))
56 colnames(yhat$st2) <- c("truth", "prob.good")
57
58 ##### Stacking Model 3: Gradient Boosting, all base learners #####
59 set.seed(2610)
60
61 control <- trainControl(method="repeatedcv", number=10,
62                          repeats=3, savePredictions=TRUE,
63                          classProbs=TRUE)
64 algorithms <- c('glm', 'rpart', 'rf', 'nnet', 'xgbLinear')
65 base <- caretList(customer~., data=train, trControl=control,
66                  methodList=algorithms)
67
68 # Stacker
69 stackControl <- trainControl(method="repeatedcv", number=10,
70                              repeats=3, savePredictions=TRUE,
71                              classProbs=TRUE,
72                              summaryFunction = twoClassSummary)
73 model_lib$st3 <- caretStack(base, method="gbm", metric="ROC",
74                             trControl=stackControl)
75
76 # Prediction on test dataset
77 yhat$st3          <- matrix(c(rownames(test),
78                              test$customer,
79                              1 - predict(model_lib$st3,
80                              newdata=test,
81                              type="prob")),
82                              ncol = 3)
83 yhat$st3 <- apply(yhat$st3, 2, function(x) as.numeric(x))
84 colnames(yhat$st3) <- c("id", "truth", "prob.good")
85
86
87 ##### Stacking Model 4: Logistic Regression, all base learners #####
88 set.seed(2610)
89
90 # Stacker
91 model_lib$st4 <- caretStack(base, method="glmnet",
92                             metric="ROC", trControl=stackControl)
93
94 # Prediction on test dataset
95 yhat$st4          <- matrix(c(rownames(test),
96                              test$customer,
97                              1 - predict(model_lib$st4,
98                              newdata=test,
99                              type="prob")),
100                              ncol = 3)
101 yhat$st4 <- apply(yhat$st4, 2, function(x) as.numeric(x))
102 colnames(yhat$st4) <- c("id", "truth", "prob.good")

```

References

- Athey, S. (2017). The impact of machine learning on economics. In *Economics of Artificial Intelligence*. University of Chicago Press.
- Basel Committee on Banking Supervision (2004). International Convergence of Capital Measurement and Capital Standards. URL: <https://www.bis.org/publ/bcbs107.pdf>. (Accessed: 03.07.2018).
- Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.
- Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2):123–140.
- Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.
- Brown, G., Wyatt, J., Harris, R., and Yao, X. (2005). Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20.
- Dheeru, D. and Karra Taniskidou, E. (2017). UCI machine learning repository. URL: <http://archive.ics.uci.edu/ml>. (Accessed: 03.07.2018).
- Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156. Bari, Italy.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.
- Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378.
- Güneş, F. (2017). Why do stacked ensemble methods win data science competitions? URL: <https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions>. (Accessed: 03.07.2018).
- Güneş, F., Wolfinger, R., and Tan, P.-Y. (2017). Stacked Ensemble Models for Improved Prediction Accuracy (*SAS paper*).
- Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182.
- Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36.

- Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844.
- Jacobson, T., Lindé, J., and Roszbach, K. (2006). Internal ratings systems, implied credit risk and the consistency of banks’ risk classification policies. *Journal of Banking & Finance*, 30(7):1899–1926.
- Kaggle. URL: <https://www.kaggle.com/>. (Accessed: 05.07.2018).
- Li, M., Andersen, D. G., Smola, A. J., and Yu, K. (2014). Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27. NIPS.
- Surowiecki, J. (2005). *The wisdom of crowds*. Anchor.
- Wager, S. and Athey, S. (in press). Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*.
- Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.
- Yu, L., Wang, S., and Lai, K. K. (2008). Credit risk assessment with a multi-stage neural network ensemble learning approach. *Expert systems with applications*, 34(2):1434–1444.
- Zhu, Y., Xie, C., Wang, G., and Yan, X. (2017). Comparison of individual, ensemble and integrated ensemble machine learning methods to predict china’s sme credit risk in supply chain finance. *Neural Computing and Applications*, 28(1):41–50.