Predicting Slaughter Weight in Pigs with Regression Tree Ensembles

Alsahaf, A.; Azzopardi, G.; Ducro, B.; Veerkamp, R. F.; Petkov, N.

Published in:
Applications of Intelligent Systems

DOI:
10.3233/978-1-61499-929-4-1

Publication date:
2018

Applications of Intelligent Systems
N. Petkov et al. (Eds.)
IOS Press, 2018

doi:10.3233/978-1-61499-929-4-1

Predicting Slaughter Weight in Pigs with Regression Tree Ensembles

A. Alsahaf[1], G. Azzopardi[1], B. Ducro[2], R. F. Veerkamp[2], N. Petkov[1]
[1]*University of Groningen, Johann Bernoulli Institute of Mathematics and Computer Science, Groningen, The Netherlands*
[2]*Wageningen University & Research, Wageningen, The Netherlands*

Abstract. Domestic pigs vary in the age at which they reach slaughter weight even under the controlled conditions of modern pig farming. Early and accurate estimates of when a pig will reach slaughter weight can lead to logistic efficiency in farms. In this study, we compare four methods in predicting the age at which a pig reaches slaughter weight (120 kg). Namely, we compare the following regression tree-based ensemble methods: random forest (RF), extremely randomized trees (ET), gradient boosted machines (GBM), and XGBoost. Data from 32979 pigs is used, comprising a combination of phenotypic features and estimated breeding values (EBV). We found that the boosting ensemble methods, GBM and XGBoost, achieve lower prediction errors than the parallel ensembles methods, RF and ET. On the other hand, RF and ET have fewer parameters to tune, and perform adequately well with default parameter settings.

Keywords. random forest, XGBoost, ensemble learning, gradient boosting, pigs, animal production.

1. Introduction

The variation of the age at which pigs reach slaughter weight has a big impact on commercial pig farming. For instance, feed cost – the highest cost in pig production – increases when slow growing pigs do not reach slaughter weight in time and must be kept for longer periods in fattening units. Labor and farm logistic costs also increase for the same reason. Moreover, if farmers are forced to supply pigs of different sizes, pork processors have to incur significant losses [1]. Therefore, early estimation of age at slaughter weight is important not only to farmers, but also to other key players in the pig supply chain.

Traditional methods that describe growth in pigs rely on statistical regression curves (e.g. linear, logistic, Gompertz, Von Bertalany) [13]. These methods use time-series data of weight measurements to fit a curve that gives a descriptive overview of how a group of pigs grow in time. However, they do not estimate the variation between individual pigs of the same group. In practice, size or growth rate estimation in farms is done using visual inspection alone.

In order to have a good estimate of future growth, different variables that affect growth should be taken into account, like sex, past weight measurements, farm conditions, genetics, litter characteristics and health. In modern pig production systems, many of these variables are recorded for individual pigs. Hence, a suitable method that combines the predictive ability of these variables is needed.

In this study, we compare four tree-based ensemble regression methods in the task of predicting age (in days) at the slaughter weight of 120 kg, namely, random forest (RF) [4], extremely randomized trees (ET) [10], gradient boosted machines (GBM) [8], and XGBoost [5]. These methods are capable of exploiting linear and non-linear dependencies between the output and the predictors. This leads them to perform favourably compared to linear models, which are commonly used in animal science applications. Additionally, they provide internal measures of feature importance that increase the interpretability of the resulting models, and consequently, their utility to the end user.

The strength of ensemble methods comes from combining weak regression or classification models to obtain models that yield better predictions than their constituents. In parallel ensembles, like RF and ET, each of the base models is trained independently on the available training data, while relying on heuristics, like bagging [3] or feature subsampling, to ensure the models vary from each other. Conversely, boosting, or sequential ensembles train on the data in a stage-wise manner, by using the output of the current stage in the training of the next one.

Gradient boosted methods, like GBM and XGBoost, are a class of boosting methods. These methods gained popularity in recent years due to their success with predictive tasks on heterogeneous tabular data, analogously to the success of convolutional neural networks with visual predictive tasks.

We highlight the main differences between the gradient boosted ensembles and the parallel ensembles in terms of predictive performance, computational efficiency, and ease of use. The remainder of the paper is organized as follows: First, a description of the used data is given, followed by a description of the compared methods. Then, regression results using default hyper-parameters and tuned hyper-parameters are presented. Finally, a discussion of the results is given, along with recommendations regarding the case example of pig slaughter age prediction, and other potential applications in livestock science.

2. Materials and methods

2.1. Data

The data used in this study consists of different phenotypic and genetic features of purebred pigs, raised in three farms in the Netherlands in the timespan between September 2013 and January 2017. The available features form a feature matrix $X_{n\times m}$, where n, the number of pigs, is equal to 32979, and m, the number of features, is equal to 28. To distinguish phenotypic features from genetic ones, we denote the former $X^{ph}_{n\times m^{ph}}$ (m^{ph}=20), and the latter $X^{EBV}_{n\times m^{EBV}}$ (m^{EBV}=8). For simplicity, we omit the subscripts from this point forward. Examples of the included phenotypic features are sex, weight measurements (at birth and at the start of the fattening period), and litter/sow characteristics like parity, gestation length, and number of born piglets. The genetic features are in the form of Estimated Breeding Values (**EBVs**) [12].

The output vector $Y_{n\times 1}$ contains the age in days at which each pig reaches 120 kilograms, which is the preferred slaughter weight in Western markets [2].

2.2 Random forest and Extra-Trees

Random forest (RF) [4] and extremely randomized trees, or Extra-Trees (ET) [10] belong to a class of ensemble tree methods in which the individual trees of the ensemble are trained independently. Different heuristics are used to introduce randomness and differentiate the individual trees from each other. Random forest does that through the use of tree bagging [3], in which each tree is trained on a random sample of the training set with replacement; and random sampling from the feature space without replacement at each tree node in the ensemble. Compared to a single decision tree, this leads to a reduction of variance without increasing bias [4].

The RF algorithm for regression has the following steps: i) Drawing M bagged samples from the training set to grow M trees; ii) Sampling p variables form the feature vector at each node in each tree, and selecting the optimal split at each node until every tree is fully grown or a stopping criterion is met; iii) Computing the final prediction as the average prediction of M trees.

The ET algorithm follows the same procedure, with two differences. First, each tree is trained on the full training set instead of a bagged sample. Second, instead of finding the optimal split from p features, a random split is chosen from each feature, then the best among the p resulting splits is selected. Consequently, this makes ET more computationally efficient than RF, because optimal split finding is the most time-critical step of the latter.

The main advantage of these algorithms, when compared to the gradient boosted algorithm introduced in the next section, is that they have fewer hyper-parameters to tune. In this study, we tune three hyper-parameters, the number of trees M in the ensemble, the number of features p sampled at each node, and the minimum number of samples to split, n_{min}. The latter controls the depth of the trees, ensuring that they are not fully grown, and thus preventing over-fitting.

2.3 Gradient boosted machines and XGBoost

Boosting describes methods that combine weak learners into a single strong learner in an iterative manner, whereby information from one step is carried over to the next. An early example of such methods is the AdaBoost classification algorithm [7], which uses an observation weighting scheme that forces subsequent learners to focus on observations that previous learners misclassified.

Gradient boosting is a framework that generalizes the idea of boosting by formulating the predictive problem as an optimization of a suitable differentiable loss function. This loss function is then solved iteratively with a greedy optimization strategy. For a regression problem, a suitable loss function could be the squared error between the true output values and their predictions. If that loss function is chosen, and the base learners are constrained to the class of regression trees, the solution to the problem becomes fitting a regression tree at the current iteration on the residuals from the previous one.

Due to the success of gradient boosted trees, there has been a recent surge of highly efficient and scalable software implementation of the algorithm. One such example is XGBoost (others include CatBoost [6], and LightGBM [11]). Described by its authors as a scalable, sparsity-aware tree boosting system, XGBoost has gained a lot of

popularity since its inception, due to achieving state-of-the-art performance on several benchmark prediction problems. Most notably, it has been part of the winning solutions to several data science competitions, such as Kaggle and and KDD cups [14].

XGBoost differs from the standard implementation of GBM in one major way. Whereas GBM minimizes a cost function which only pertains to the prediction error (Eq.1), while leaving regularization to heuristics (e.g. limiting tree depth), XGBoost builds trees that explicitly deal with regularization. It does so by having a cost function that includes a loss component, as well as a regularization component that controls the complexity of the trees (Eq. 2). This leads to building efficient trees that utilize a specialized node splitting score which automatically adjusts tree depth and leaf values to ensure regularization.

Additionally, XGBoost contains strategies that speed up learning for very large datasets. This is done by exploiting sparsity when it is present in the inputs; and through a split finding algorithm which efficiently finds approximates to the optimal splits. Finally, XGBoost optionally utilizes many of the techniques introduced in other tree ensemble algorithms, like sampling from the input feature space [4], and shrinkage [9].

Due to the wider choice of options and hyper-parameters, tuning an XGBoost model is less trivial than in the case of RF, ET, and standard GMB. In the next section, we demonstrate how all four algorithms perform in pig slaughter weight prediction with default hyper-parameters, and how much improvement they show after partial tuning of their hyper-parameters.

$$obj(\Theta) = L(\Theta) \quad (1)$$

$$obj(\Theta) = L(\Theta) + \Omega(\Theta) \quad (2)$$

where Θ refers to the model parameters, $obj(\Theta)$ is the objective function that the gradient boosted model tries to minimize, $L(\Theta)$ is a loss term that measures how well the model fits to the training data, and $\Omega(\Theta)$ is a regularization terms which controls the complexity of the model.

3. Results

3.1 Default settings

To test the efficacy of the four methods, we compare them under two conditions. First, using their default hyper-parameters, as prescribed by either the authors of the algorithms or the software packages that implement them. Then, by tuning those parameters by a grid search. All methods are evaluated with 5-fold cross-validation, and the average performance on the test sets is presented, measured by R^2, mean absolute error, and average training time. Each method is trained and evaluated with feature matrices X, X^{ph}, and X^{EBV} separately, to assess the difference in predictive power between phenotypes and EBVs. The default parameters are given in Table 1, and the performances of the corresponding models are given in Table 2.

Table 1. The hyper-parameters that are controlled in each method, and their corresponding default* values.

Method	Hyper-parameters and default values	Description
RF and ET	M 500	Number of trees
	p $n_{feature}$	Number of features sampled randomly at each node
	$n_{min} = 2$	Minimum number of observations for splitting a node
GBM	M = 500	Number of trees
	p $n_{feature}$	Number of features sampled randomly at each node
	n_{min} 2	Minimum number of observations for splitting a node
	l_r 0.1	Learning rate
XGBoost	M = 500	Number of trees
	l_r 0.1	Learning rate
	gamma 0	minimum loss reduction required to make a split
	$\alpha_{reg} = 0$	L1 regularization term on the observation weights
	λ_{reg} =1	L2 regularization term on the observation weights
	% features 1	Ratio of features used for building a tree

* Default values for all hyper-parameters except M are taken from the scikit-learn implementation of RF, ET, and GBM; and the official software library of XGBoost. The number of trees, M, is unified for all methods to make the comparison fair.

Table 2. The performance (evaluated in 5-fold cross-validation) of the four algorithms with default parameters, measured by R^2, mean absolute error (MAE), and average time (in seconds) to train a fold (t_{tr}). The performance is evaluated independently on feature matrices X^{ph}, X^{g}, and X.

Method	Feature matrix								
	X^{ph}			X^{EBV}			X		
	R^2	MAE	t_{tr}	R^2	MAE	t_{tr}	R^2	MAE	t_{tr}
RF	0.6171	8.7895	102.2844	0.3294	11.8745	61.8950	0.6275	8.6711	174.3972
ET	0.6191	8.7096	51.5155	0.1709	13.1560	20.9858	0.6304	8.6057	71.9734
GBM	0.6308	8.6377	8.6220	0.2492	12.6441	4.3684	0.6425	8.4794	12.5490
XGBoost	0.6310	8.6361	4.5714	0.2464	12.6723	2.9744	0.6434	8.4720	7.0729

3.2 Tuning hyper-parameters

The hyper-parameter common to all four methods is the number of trees. In RF and ET, this refers to the number of trees trained independently in the ensemble, while in the gradient boosted methods, GBM, and XGBoost, it refers to the number of boosting rounds. In the first step of tuning the hyper-parameters, we select an appropriate value for the number of trees by evaluating how the performance changes as the number of trees is increased, while keeping other hyper-parameters at their default values. Then, we select a value above which the performance does not improve.

Figure 1 shows that an appropriate value for RF and ET can be set at 500 trees, as any higher value would only increase computational time without impacting performance. On the other hand, GMB and XGBoost appear to benefit from larger ensembles, i.e., more boosting rounds. We therefore set the number of trees in both methods to 3000.

The next step in tuning the hyper-parameters is to fix the number of trees and perform a grid search over other relevant hyper-parameters. Those parameters are given in Table 1. The performance of the models with the tuned hyper-parameters is given in Table 3.

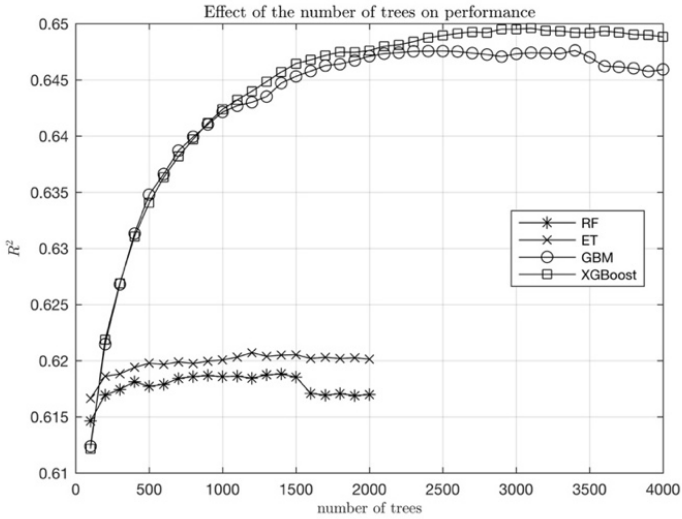

Figure 1. The average 5-fold cross validated R^2 on test sets as the number of trees is increased, using the algorithms random forest (RF), Extra-Trees (RT), gradient boosted machines (GBM), and XGBoost.

Table 3. The performance (evaluated in 5-fold cross-validation) of the four algorithms with tuned parameters, measured by R^2, mean absolute error (MAE), and average time (in seconds) to train a fold (t_{tr}). The performance is evaluated independently on feature matrices X^{ph}, X^{g}, and X.

Method	**Feature matrix**								
	X^{ph}			X^{EBV}			X		
	R^2	MAE	t_{tr}	R^2	MAE	t_{tr}	R^2	MAE	t_{tr}
RF	0.6223	8.7520	24.6565	0.3156	11.9978	22.4271	0.6377	8.5647	36.3896
ET	0.6233	8.7396	12.4734	0.3575	11.6172	9.0036	0.6381	8.5561	15.1522
GBM	0.6685	7.9595	47.7744	0.3426	11.7856	30.9345	0.6739	7.9404	81.2535
XGBoost	0.6672	8.0306	23.6216	0.3376	11.8287	18.1496	0.6742	7.9737	35.8121

3.3 Feature importance

In the scikit-learn implementations of RF, ET, and GBM, the importance of a feature is calculated by accumulating the splitting score caused by that feature whenever it is selected to split a node. In the XGBoost software package, three different feature importance scores are available, each having a different definition and interpretation: 'weight': is the number of times the feature is selected to split a node in all of the ensemble, 'gain': is the total splitting score of the feature, similar to scikit-learn, and 'cover': is the the number of samples covered by the splits caused by that feature.

Figure 2 shows the feature importance scores relative to each other using the aforementioned metrics. The feature importance scores are derived from training the models with all samples, and using the entire feature matrix X. The scores from each method are normalized such that the highest feature has a score of one.

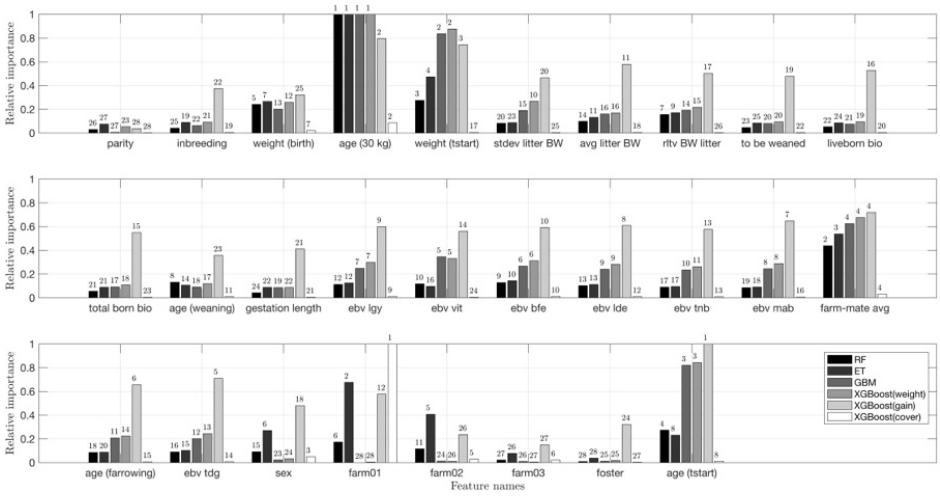

Figure 2. The normalized feature importance scores (bar height) and ranking (number above bar) using the feature importance metrics of RF, ET, GBM, and XGBoost. See the Appendix for feature description.

4. Discussion

The regression outcomes prior to tuning the hyper-parameters (Table 2) show that GBM and XGBoost only have a small advantage in prediction accuracy over RF and ET, with the best performing method, XGBoost, having a mean absolute error that is lower than the worst performing method, RF, by just 0.2 days. On the other hand, GBM and XGBoost show a big advantage over RF and ET in computational efficiency, with XGBoost having an average training time that is an order of magnitude shorter than that of ET, which in turn is faster than RF.

After tuning the hyper-parameters, XGBoost and GBM show a bigger advantage in prediction accuracy over RF and ET, with a difference in mean absolute error of approximately 0.6 days.

The regression outcomes further suggest that nearly all of the predictive power comes from the phenotypic features, X^{ph}. Estimated breeding values, X^{EBV}, contain limited predictive power on their own. However, when combined with X^{ph}, they do not significantly improve performance. This indicates redundancy between the features of X^{EBV} and those of X^{ph}.

Inspecting the feature importance scores and relative rankings (Fig. 2) allows further insight about the relevance of different features in predicting slaughter weight. With the exception of the XGBoost 'cover' metric, all metrics rank the following features in the topmost positions: 'age (30 kg)', 'weight (tstart)', 'age (tstart)', and 'farm-mate avg'. The first feature is the age of the animal at 30 kg, the second and third are the weight and age of the animal at the start of the finishing stage, respectively, and the last is the average age at 120 kg of farm/sex/line-mates in the preceding three months. Furthermore, the feature ranks suggest that the most important features from X^{EBV} are 'ebv vit' and 'ebv bfe', the estimated breeding values related to vitality and back fat thickness, respectively.

5. Conclusion

In this work, we demonstrate the utility of regression tree ensembles in predicting a future phenotype of an animal, namely, the age at 120 kg in pigs. Due to being based on decision trees, these methods are well-suited for such applications, which often contain different types of input variables, such as phenotypic and genetic variables.

By comparing four regression ensemble algorithms, we echo the conclusions of the machine learning community that XGBoost has a significant performance advantage over parallel ensemble methods, as well as the scikit-learn implementation of gradient boosting. Random forest and Extra-Trees, however, have an advantage of being conceptually simpler, and requiring less tuning from the user.

References

[1] Apichottanakul, A., Pathumnakul, S., and Piewthongngam, K., 2012. The role of pig size prediction in supply chain planning. biosystems engineering 113 (3): 298-307.
[2] Boland, M. A., Foster, K. A., Preckel, P. V., Schinckel, A. P., 1996. Analyzing pork carcass evaluation technologies in a swine bioeconomic model, Journal omf production agriculture 9 (1): 45-49.
[3] Breiman, L., 1996. Bagging predictors. Machine learning 24(2): 123-40.
[4] Breiman, L., 2001a. Random forests. Machine learning 45 (1): 5-32.
[5] Chen, T. and Guestrin, C., 2016, August. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.
[6] Dorogush AV, Ershov V, Gulin A. CatBoost: gradient boosting with categorical features support.
[7] Freund, Y. and Schapire, R.E., 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, *55*(1), pp.119-139.
[8] Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp.1189-1232.
[9] Friedman, J.H., 2002. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), pp.367-378.
[10] Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Machine learning. 2006 Apr 1;63(1):3-42.
[11] Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu TY. LightGBM: A highly efficient gradient boosting decision tree. InAdvances in Neural Information Processing Systems 2017 (pp. 3149-3157).
[12] Kor, O. and Van der Waaij, L, 2014. Textbook Animal Breeding: Animal Breeding Andgenetics for BSc Student*s*. Centre for Genetic Resources and Animal Breeding and Genomics Group, Wageningen University and Research Centre.
[13] Luo, J., Lei, H., Shen, L., Yang, R., Pu, Q., Zhu, K., Li, M., Tang, G., Li, X., Zhang, S., 2015. Estimation of growth curves and suitable slaughter weight of the liangshan pig, Asian-Australasian journal of animal sciences 28 (9) 252
[14] https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

Appendix

Table 4. The full list of features in X^{ph}, X^{EBV}, and the output Y.

Feature name	Description (unit)	Type	Range	Mean $\pm$ std
parity	Parity number of biological mother	X^{ph}	1 - 13	2.73 ± 1.63
weight (birth)	Weight at birth (g)	X^{ph}	330 - 3250	1380 ± 298
age (30 kg)	Age at 30 kg (days)	X^{ph}	48.9 - 115.3	76.44 ± 8.09
age (tstart)	Age at the start of the finishing phase (days)	X^{ph}	39 - 168	77.54 ± 11.44

farm-mate avg	Age at 120 kg of farm-line-sex mates in last 3 months (days)	X^{ph}	156 - 202	182.19 ± 10.97
age (farrowing)	Age of biological mother at farrowing (days)	X^{ph}	313 - 2119	616.48 ± 243.88
age (weaning)	Age at weaning (days)	X^{ph}	1 - 63	23.99 ± 4.57
weight (tstart)	Weight at the start of the finishing phase (kg)	X^{ph}	15 - 50	31.21 ± 7.07
stdev litter BW	Std. deviation in birth weight in biological litter	X^{ph}	0 - 1036	279.26 ± 80.31
avg litter BW	Average birth weight in biological litter (g)	X^{ph}	600 - 2740	1299.28 ± 211.61
rltv BW litter	Relative birth weight of animal compared to littermates (g)	X^{ph}	-1080 - 1160	80.79 ± 230.57
to be weaned foster	Number of piglets to be weaned by the foster mother	X^{ph}	0 - 38	13.59 ± 2.89
liveborn bio	Number of born alive piglets in the biological litter	X^{ph}	1 - 28	14.23 ± 3.28
total born bio	Number total born piglets in the biological litter	X^{ph}	1 - 30	15.53 ± 3.44
gestation length	Gestation length of biological dam (days)	X^{ph}	108 - 123	115.18 ± 1.59
inbreeding	Inbreeding coefficient	X^{EBV}	0 - 0.26	0.0178 ± 0.0180
sex	Female or male	X^{ph}	Binary	-
farm01	Farm of birth – farm 01	X^{ph}	Binary	-
farm02	Farm of birth – farm 02	X^{ph}	Binary	-
farm03	Farm of birth – farm 03	X^{ph}	Binary	-
foster	Fostered by biological or foster dam.	X^{ph}	Binary	-
ebv lgy	Breeding value for sow longevity [parent average]	X^{EBV}	-0.79 - 1.12	0.05 ± 0.24
ebv vit	Breeding value for piglet vitality [current EBV]	X^{EBV}	-11.9 - 12.6	0.14 ± 3.17
ebv bfe	Breeding value for back fat thickness [parent average]	X^{EBV}	-3.69 - 2.4	-0.28 ± 0.89
ebv lde	Breeding value for loin depth thickness [parent average]	X^{EBV}	-4.83 - 5.98	0.52 ± 1.55
ebv tnb	Breeding value for total number of born piglets [parent average]	X^{EBV}	-2.25 - 2.69	-0.04 ± 0.59
ebv mab	Breeding value for mothering ability [parent average]	X^{EBV}	-6.58 - 4.90	0.08 ± 1.39
ebv tdg	Breeding value for daily gain [calculated by quarter]	X^{EBV}	31.22 - 39.79	35.21 ± 1.45
age (120 kg)	Standardized age at 120 kg	Y	120.30 - 265.60	182.97 ± 18.48