SUPPLEMENTARY MATERIAL TO "EXPLAINABLE MACHINE LEARNING ALGORITHMS TO PREDICT GLASS TRANSITION TEMPERATURE"

Edesio Alcobaça,* Saulo Martiello Mastelini,* Tiago Botari,* Bruno Almeida Pimentel,
Daniel Roberto Cassar, André Carlos Ponce de Leon Ferreira de Carvalho and Edgar Dutra Zanotto
Institute of Mathematics and Computer Sciences
University of São Paulo, São Carlos, Brazil
and
Department of Materials Engineering
Center for Research, Technology, and Education in Vitreous Materials
Federal University of São Carlos, São Carlos, Brazil
{bapimentel, andre}@icmc.usp.br, {edesio, mastelini}@usp.br
{tiagobotari,daniel.r.cassar}@gmail.com, dedz@ufscar.br

1 Statistics of the data

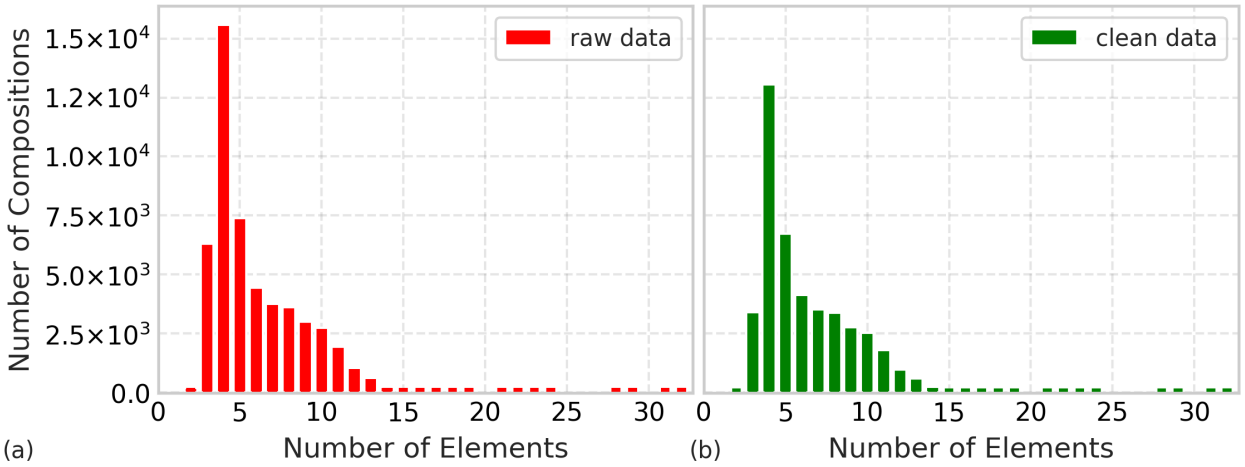

Figure 1: Histogram showing number of different elements in the compositions of the T_g dataset. Raw data is presented in (a) and data without duplication in (b).

∗ These authors contributed equally to this work.

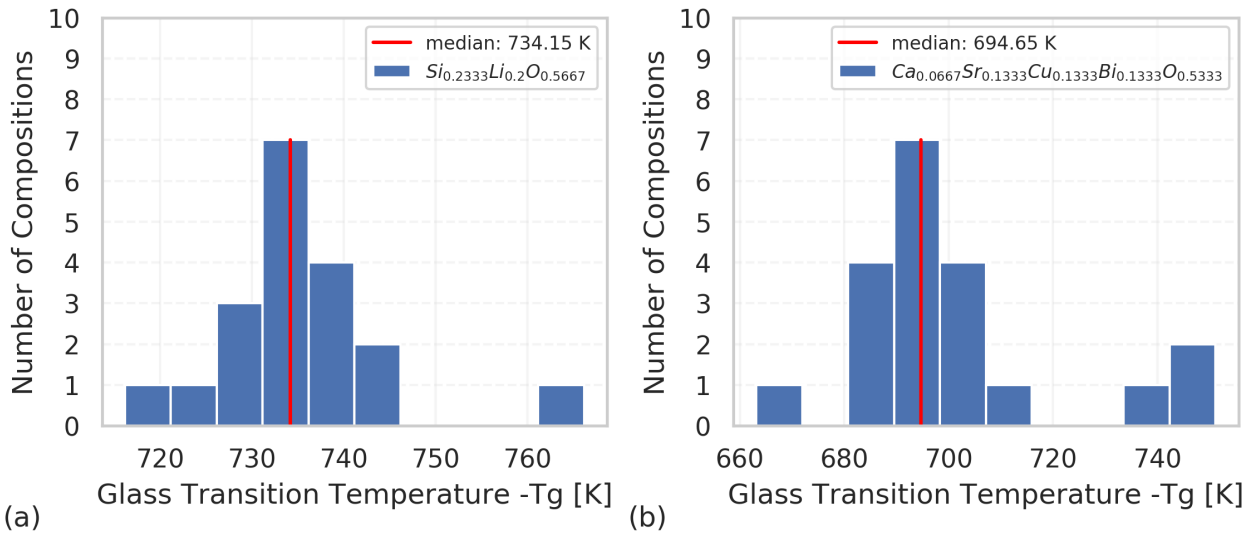

Figure 2: Histogram of T_g for duplicated instances for chemical composition: (a) $Si_{0.2333}Li_{0.2}O_{0.5667}$ and (b) $Ca_{0.0667}Sr_{0.1333}Cu_{0.1333}Bi_{0.1333}O_{0.5333}$. From the duplicated instances, the T_g median was calculated obtaining (a) 734.15 K and (b) 694.65 K. Then, all duplicated instances were replaced with just one instance where the T_g equals its median.

2 Machine Learning Algorithms

In this section we described the six ML algorithms used in this work. More detail can be found in the cited references.

MLP: Based on the nervous system, Multilayer Perceptron (MLP) [1] neural networks are usually trained by the backpropagation algorithm [2]. A MLP neural network architecture has one output layer and, at least, one hidden layer. as we increase the number of hidden layers, the functionality of the MLP network increases, but there is also an increase in its complexity, which can result in overfitting. MLP neural networks can be applied to classification and regression problems. When applied to regression problems, they produce a real value as output.

SVR: Support Vector Regression (SVR) [3] is a variation of SVM for regression tasks. SVM, in turn, is based on the statistical learning theory [4] and fits hyperplanes with large margins to the training data. When applied to classification tasks, SVM looks for a hyperplane that best separates the training data, with the largest separation margin between the decision border and a set of support vectors extracted from the training data. It is originally restricted to linearly separable problems. However, a non-linearly separable feature space can be transformed to a larger space using kernel functions. In this new representation, the problem becomes linearly separable. The most often used kernel functions are: Linear, Polynomial and Gaussian.

CatBoost: The Categorical Boosting (CatBoost) [5] algorithm is based on gradient boosting [6] (a powerful ensemble technique used in ML). It is supported by theoretical studies that explain how strong predictors can be built by iterative combining weak predictive models (base predictors). This occurs via a greedy procedure similar to gradient descent in a function space. Most of the popular implementations of gradient boosting, including the CatBoost, use decision trees, here regression trees, as base predictors.

k-NN: The k-Nearest Neighbors (k-NN) [7] is an instance-based or lazy learning algorithm that locally approximates the predictive function and defers all the computations until the classification of a new instance. Even though k-NN is among the simplest ML algorithms, it has presented a competitive predictive performance in several ML experiments. When the k-NN algorithm is applied to a dataset, the predicted value is defined by the label of the k closest training examples in the feature space. In regression problems, the predicted value is the average of the label value of the k closest instances.

RF: The Random Decision Forests [8] or Random Forest (RF) [9] is an ensemble learning algorithm often used for classification and regression tasks. RF works by inducing several decision trees during its training. When applied to a test set, RF outputs the mean of the predictions of the individual trees. For regression tasks, RF induces a forest of regression trees. Each tree is induced via the sample with a replacement from the training set, using an ensemble building procedure named Bagging [10]. As a consequence, RFs tend to be less prone to overfitting than single trees.

CART: The Classification and Regression Tree (CART) algorithm [11] induces predictive models represented by decision trees, in this study, regression trees. A regression tree has a tree-like shape, which is defined when the algorithm is applied to a training set. Each tree node is associated with a predictive attribute from the dataset. Each path from the tree root to a leaf is a regression rule. When applied to a test set, each instance goes through a path, from the tree root to a leaf, according to the value of its predictive attributes. The value in the leaf node is the regression tree output for the instance.

3 Evaluation Measures

In this section, we presented the evaluation measures and the equations that are used to evaluate our calculations.

RMSE: Compares the squared deviations from the expected outcomes and takes the square root from the aggregated errors, as shown in Equation 1,

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} \, . \tag{1}$$

RRMSE: Assesses the predictive performance regression algorithms, bringing the advantage of measuring the target algorithm improvements over a baseline (a predictor that always predicts the mean value of the desired output). The RRMSE can be calculated using Equation 2.

$$\text{RRMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{2}$$

RD: Measures how the predictions deviate from the expected outcome in percentage. For such, the absolute deviation of the predictions is divided by the measured values for each object, as shown in Equation 3.

$$\text{RD} = \frac{100}{N} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{y_i} \, . \tag{3}$$

R^2: Proportion of the variance in the dependent variable that is predictable from the independent variable(s). R^2 can be calculated using Equation 4.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \, . \tag{4}$$

4 Hyperparameters Tuned

In this supplementary session we describe the set and range of hyperparameters used during the tuning of the regression algorithms.

Table 1: Hyperparameters tuned for the Catboost algorithm along with their best values found

Hyperparameter	Description	Range	Best value
iterations	The maximum number of models to be trained.	$[100, 1000]$	949
learning_rate	Learning rate used for reducing the gradient step.	$[0.01, 0.40]$	0.3852
depth	The maximum depth for the tree models.	$[1, 16]$	14
l2_leaf_reg	The $L2$ regularization coefficient, which is employed in the leaves' responses calculation.	$[0, 7]$	4.1899
random_strength	Variance range for a normal random variable which is added to the split scores, aiming at decreasing overfitting. The mentioned variance automatically decreases during the training.	$[0, 1]$	0.1309
bagging_temperature	Settings for the Bayesian bootstrap, *i.e.*, the weights for sampling each instance. When set to 1, the weights are sampled from an exponential distribution; meanwhile, setting this parameter to 0 makes all the weights become 1 (uniform).	$[0, 1.5]$	0.0107
border_count	Number of splits to use for numerical attributes.	$[128, 254]$	142

Table 2: Hyperparameters tuned for the k-NN algorithm along with their best values found

Hyperparameter	Description	Range	Best value
n_neighbors	Number of neighbors to consider in the algorithm.	$[1, 1000]$	6
weights	Strategy adopted to combine the neighbor information: either *"uniform"* or *"distance"*. The former option means that simple arithmetic mean will be performed to combine the neighbors' target values; the latter option indicates that the combination will use weights inversely proportional to the distance of the neighbor to the example of interest.	–	*distance*

Table 3: Hyperparameters tuned for the design and training of the MLP network, along with their best values found

Hyperparameter	Description	Range	Best value
hidden_layer_sizes	MLP architecture (layer and neurons per layer) used. We trained MLPs with at most three hidden layers.	$[0, 100]$	$(76, 99, 95)$
solver	Solver used for the weight optimization. The evaluated values were: "lbfgs" (an optimizer in the family of quasi-Newton methods), "sgd" (Stochastic Gradient Descent), and "adam".	–	*adam*
activation	Activation functions used for the artificial neurons. We evaluated the following possibilities: "logistic", "tanh" (hyperbolic tangent), and "relu".	–	*relu*
alpha	A regularization parameter ($L2$ penalty). We evaluated the following values for this parameter: $10^{-5}, 10^{-4}$, and 10^{-3}.	–	10^{-5}
learning_rate	Strategy for updating the learning rate. We evaluated two possibilities: "constant" (the learning rate is kept constant) and "adaptive" (maintains the learning rate constant while the training error is decreasing; every time two epochs fail to decrease the error, the current learning rate is divided by 5).	–	*constant*
learning_rate_init	Initial learning rate.	$[0.001, 0.1]$	0.0239
batch_size	Size of mini-batches for stochastic optimizers. We evaluated three possible values: $\{200, 500, 1000\}$.	–	200
max_iter	The maximum number of allowed iterations.	$[200, 1000]$	986
momentum	Momentum for the gradient descent optimizer.	$[0, 1]$	0.5032

Table 4: Hyperparameters tuned for the CART algorithm along with their best values found

Hyperparameter	Description	Range	Best value
criterion	Criterion used for node splitting. We explored two possible values, which were either *"mse"* or *"friedman_mse"*. The first option refers to the traditional mean square error (MSE), while the second one uses MSE along with the Friedman's improvement score for potential splits.	–	0.0509
min_impurity_decrease	The minimum amount of impurity a split must reduce in order to be performed. The term impurity refers to the split criterion used. The impurity reduction calculation is weighted by the number of examples lying in each induced tree branch.	–	*mse*

Table 5: Hyperparameters tuned for the RF algorithm along with their best values found

Hyperparameter	Description	Range	Best value
n_estimators	Indicates the number of trees of the ensemble.	$[500, 1000]$	933
max_features	The maximum number of features to consider at each split attempt. We considered three built-in settings for RF: *"auto"*, *"sqrt"*, and *"log2"*. The first option uses n_features, the second option $\sqrt{n}$_features, and the last one $\log_2(n$_features$)$.	–	*sqrt*

Table 6: Hyperparameters tuned for the SVR algorithm along with their best values found

Hyperparameter	Description	Range	Best value
kernel	Defines the kernel function for training the SVR model. We evaluated three possibilities, namely: *"linear"*, *"poly"* (polynomial), and *"rbf"* (Radial Basis Function).	–	*poly*
degree	Defines the degree of the polynomial function in case kernel is set to "poly".	$[2, 4]$	4
gamma	Kernel coefficient, used when *"poly"* or *"rbf"* are selected. We explored two built-in settings of sklearn's implementation: *"auto"* and *"scale"*. The former setting sets *gamma* as $\frac{1}{n\_features}$, whereas the latter one uses $\frac{1}{n\_features \cdot \sigma^2}$. In the last expression, σ^2 represents the variation of all features and examples in the training set.	–	*scale*
coef0	Independent term (intercept) for the polynomial kernel.	$[0, 100]$	4.5713
tol	Tolerance for the stopping criterion was kept constant at 10^{-4} in our experiments.	–	–
C	Penalty parameter of the error term.	$[0.01, 100]$	93.2874
epsilon	Specifies the epsilon-tube, *i.e.*, a tolerance margin around the actual desired values, for which no training loss is associated to points lying within its boundaries.	$[5, 50]$	5.7217
max_iter	The maximum number of iterations for the solver was kept constant at 10^6.	–	–

5 Scatter plot of tuned and not tuned approaches

The difference in predictive performance when using tuned and non-tuned (default) hyperparameters can be better visualized by comparing the spread of the true T_g values versus the observed predictions. To this end, we present scatter plots for each regression algorithm used. In these figures, the x-axis represents the true measured values of T_g, whereas the y-axis represents the value predicted by the predictive model.

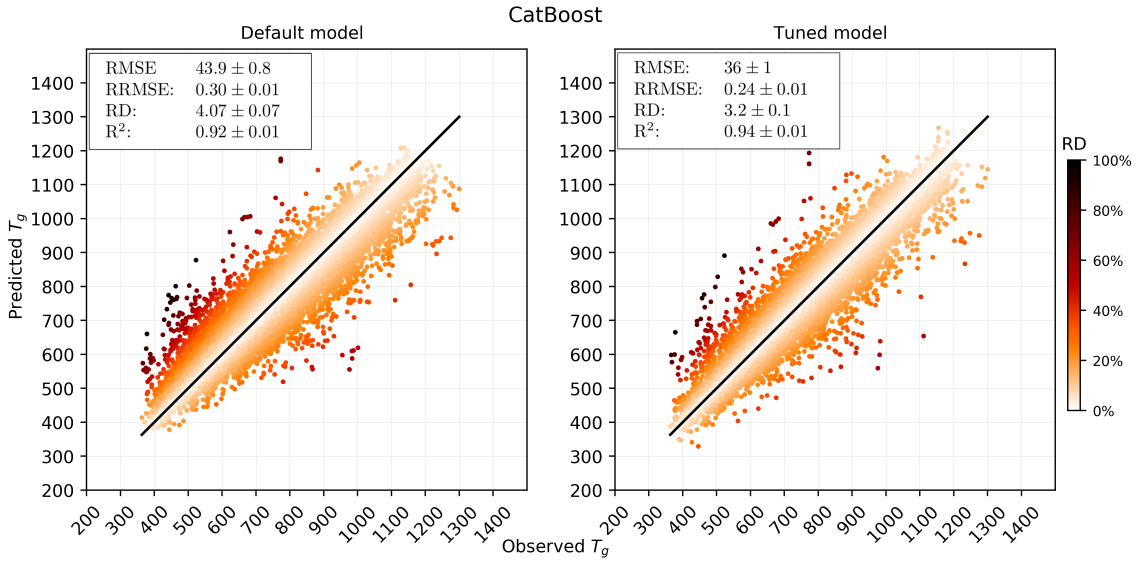

Figure 3: Predictions scattering for Catboost: default vs. tuned hyperparameters

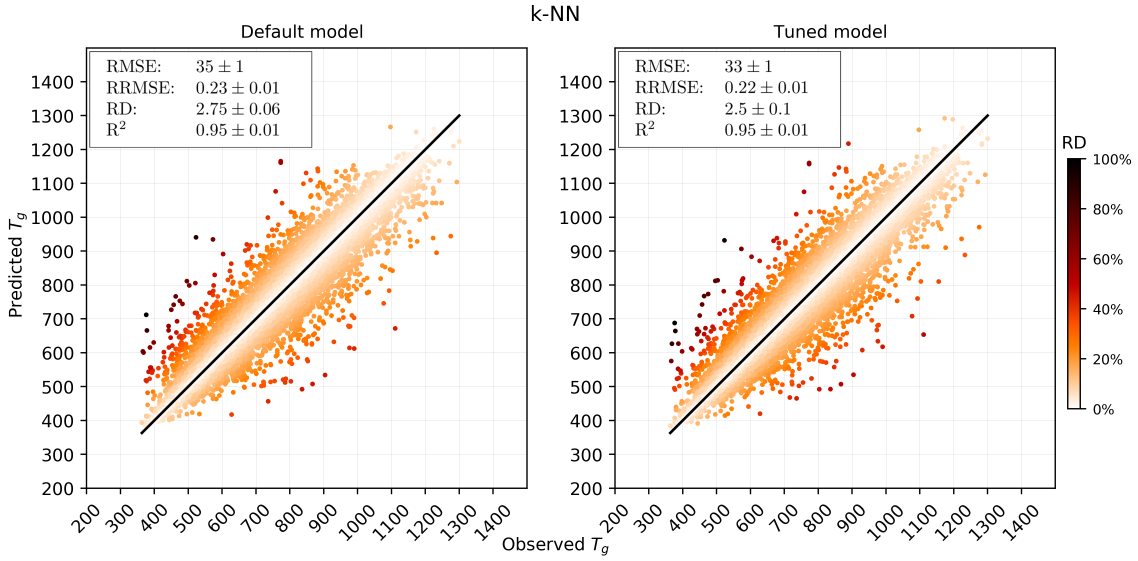

Figure 4: Predictions scattering for k-NN: default vs. tuned algorithm.

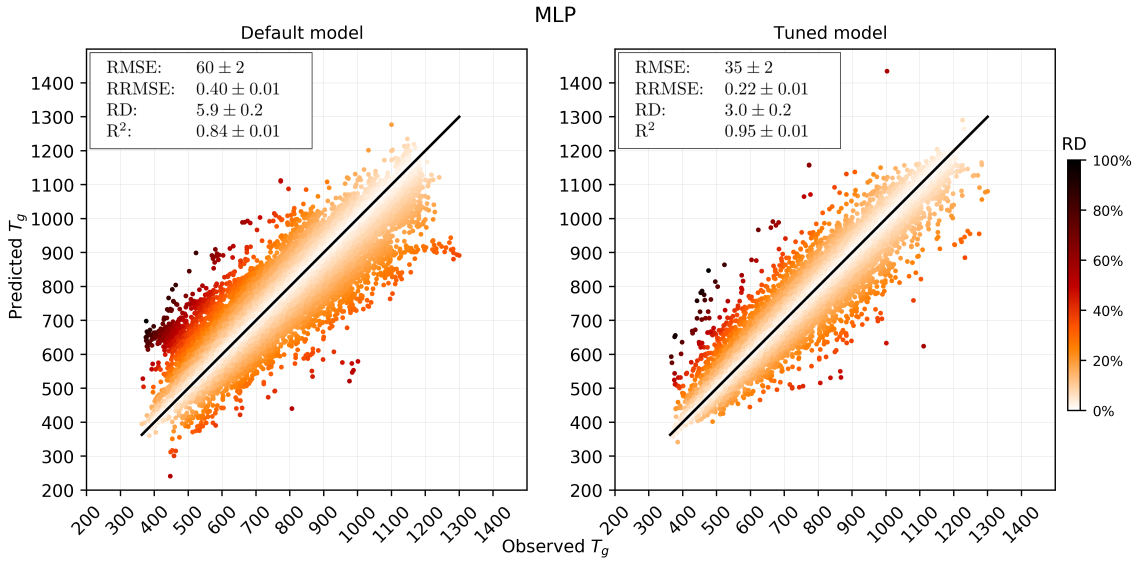

Figure 5: Predictions scattering for MLP: default vs. tuned hyperparameters

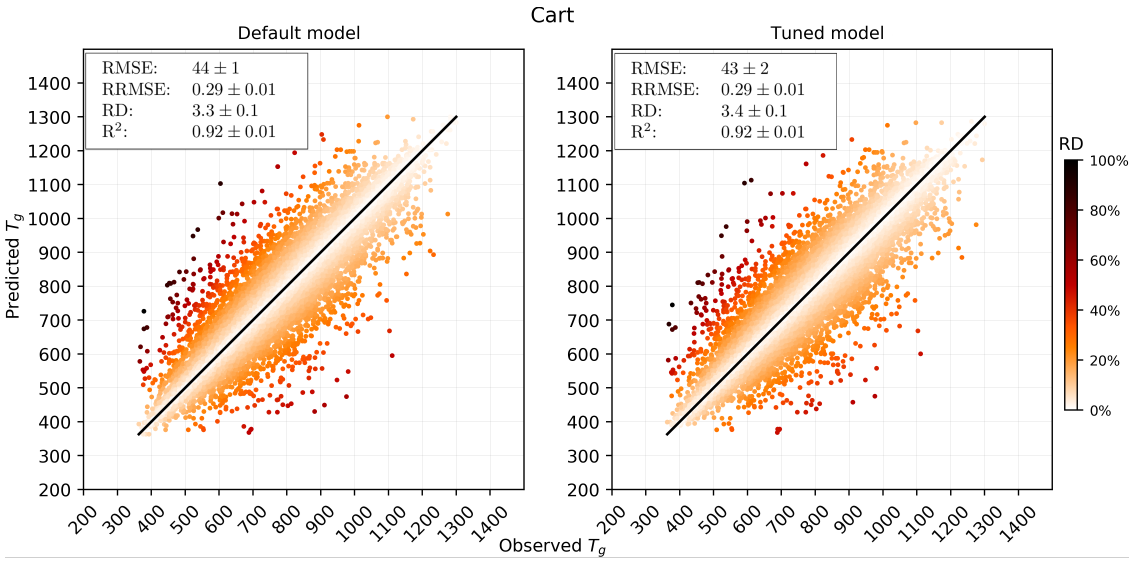

Figure 6: Prediction scattering for CART: default vs. tuned hyperparameters

6 RF Feature importance analysis

We also assess the built-in feature importance measure of RFs. This traditional measure used by the ML community is different from the one we used in our analysis, as previously stated. In a brief explanation, during the building of its model, the RF algorithm must select the predictive attributes from the dataset (features) it considers the most relevant to create each regression tree. In our experiments, the features are quantities of chemical elements. The higher the number of times a chemical element is selected to be part of a regression tree, the more "important" this chemical element considered by RF to predict the T_g value.

RF uses bootstrap samples (in the so-called Bagging aggregation ensemble approach) to induce each regression tree in the forest. For each regression tree, RF uses a set of Out-of-Bag (OOB) examples, which were not used during training, to calculate an unbiased error per tree, named Out-of-Bag Error (OOBE). The feature importance calculation measures how the forest's mean OOBE would increase when the values of each feature is randomly permuted in the OOB examples. This enables us to estimate the feature importance considering the whole ensemble.

By using the RF built-in feature importance measure, we can extract from the trained models information regarding the relation between the chemical composition and the glass transition temperature. This information can be validated by experts in the area, and in this process can also bring new insights. This analysis is presented in Figure 7 and was conducted by averaging the feature importance measured for each tuned RF from each 10-CV training partition. In this figure, the bars represent the percentage of contribution from each chemical element when predicting the T_g value using RF. The small intervals above each bar represent the standard deviation for all RF models considered. To better illustrate these results, the y-axis is shown in logarithm scale.

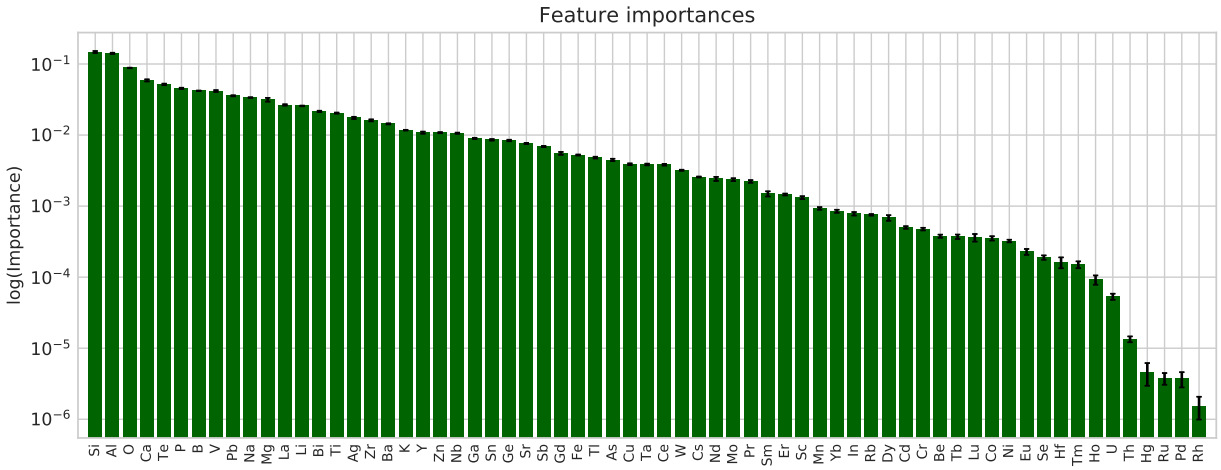

Figure 7: RF feature importance for the T_g prediction

7 Investigation of Replacement of Duplicated Glass Compositions by the T_g Median Value

In this subsection, we investigated the two best models' performance regarding the replacement of duplicated glass compositions with different T_g values. When using duplicated compositions, care should be taken concerning data leakage, *i.e.*, when the same glass composition is randomly selected by the splitting algorithm to be on the training and testing folds. We can deal with this problem by creating folds with no duplicated compositions and then adding the duplicated compositions later, thus avoiding duplicated materials in the training and test folds at the same time. However, the number of instances in each fold will be slightly different from each other.

Thus, we investigated the two best models (RF and kNN) when duplicated instances are included in the dataset. We followed the same methodology to evaluate the errors of the two models trained without duplicated instances used in Section 3 of the main text. To split the data into 10-folds, we first split the data using unique glass compositions. Next, we added the duplicated instances on each fold containing the corresponding duplicated instance. Using this strategy, we avoided data leakage between the training and test sets. Afterwards, using the parameters of the best models, we induced and evaluated the models leaving 1 fold for test and the other 9 for training. We performed this process with each fold as a test set and obtained the average over the ten folds. The evaluation metrics are presented on table 7 for RF best model and 8 the kNN best model. From the evaluation metrics, we observed only a minor increase in the errors when the models are trained using the duplicated instances.

Table 7: Measures of errors for best RF model trained over 10-folds.

Measures	**Best RF with median**	**Best RF with duplicate values**
RMSE	30 $\pm$1	30 $\pm$2
RRMSE	0.20 $\pm$0.01	0.20 $\pm$0.01
RD	2.38 $\pm$0.06	2.45 $\pm$0.06
R^2	0.96 $\pm$0.01	0.96 $\pm$0.01

We also analyzed eight different glass compositions from 2 different folds for the best RF model trained over the other 9 folds. For the 8 glass composition analyzed, we did not detect significant differences over the predicted models when the duplicated instances were included compared to models trained using the median values. The distributions of T_g and the predicted values for the model trained with duplicated and median values are shown in figure 8 and 9, respectively.

An advantage of using the median values instead of including repeated property values is the possibility of removal of outliers, besides avoiding data leakage. Therefore, we conclude that the median value approach is better than including duplicated data.

Table 8: Measures of errors for best KNN model trained over 10-folds.

Measures	Best KNN with median	Best KNN with duplicate values
RMSE	33 ±1	34 ±2
RRMSE	0.22 ±0.01	0.23 ±0.01
RD	2.5 ±0.1	2.66 ±0.07
R^2	0.95 ±0.01	0.95 ±0.01

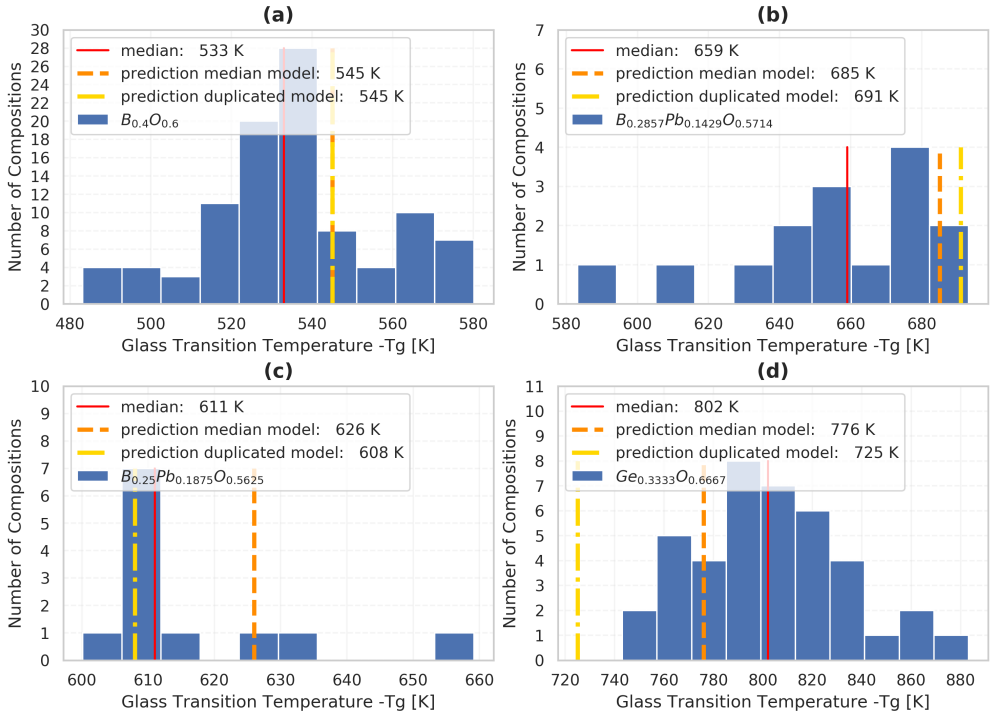

Figure 8: Histogram of T_g values over duplicated glass compositions and predictions for the best RF models when trained using duplicated values and with duplicated values replaced by the median value, fold 1.

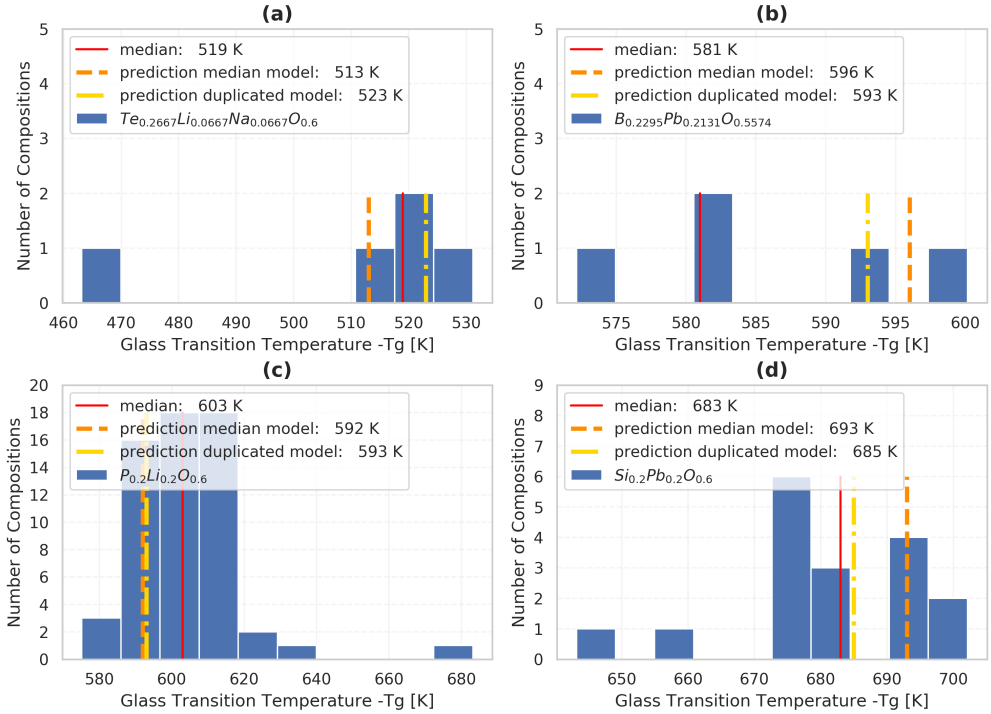

Figure 9: Histogram of T_g values of duplicated glass compositions and predictions for the best RF models when trained using duplicated values and with duplicated values replaced by the median value, fold 2.

References

[1] Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2**, 359–366 (1989).

[2] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning internal representations by error propagation. Tech. Rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985).

[3] Smola, A. J. & Schölkopf, B. A tutorial on support vector regression. *Statistics and computing* **14**, 199–222 (2004).

[4] Vapnik, V. *The nature of statistical learning theory* (Springer science & business media, 2000).

[5] Dorogush, A. V., Ershov, V. & Gulin, A. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363* (2018).

[6] Natekin, A. & Knoll, A. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics* **7**, 21 (2013).

[7] Weinberger, K. Q. & Saul, L. K. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research* **10**, 207–244 (2009).

[8] Ho, T. K. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, 278–282 (IEEE, 1995).

[9] Breiman, L. Random forests. *Machine learning* **45**, 5–32 (2001).

[10] Breiman, L. Bagging predictors. *Machine learning* **24**, 123–140 (1996).

[11] Breiman, L. *Classification and regression trees* (Routledge, 2017).