Machine learning in geoscience: facies classification with features engineering, clustering, and gradient boosting trees

Marcelo Guarido*

ABSTRACT

Facies classification is the process to determine the local rocks lithology by analyzing indirect measurements, such as well logs. Usually it is done manually by an interpreter. In this work, I am presenting an automatic method for facies classification by the use of feature engineering and gradient boosting trees. I used a set of classified well logs to train a multi-class machine learning model, and compared the predictions with both raw and processed features in a blind well. I could demonstrate that preparing the, by creating new features from the original well logs, such as their gradients, polar coordinates transformations, and clustering analysis, increased the predictions accuracy from 47% to 60%.

INTRODUCTION

During the mining and Oil&Gas well exploration, the local lithology can be determined, with high accuracy, by core analysis (extracting the rocks at different depth and analyzing their mineralogy). However, due to the high cost of such extraction, its not always an option. In those cases, indirect methods to determine the rocks types are required (Richard F. Wadleigh and Ward, 1984; Crampin, 2008).

Facies, or lithofacies, classification consists on determining a rock type, in a given depth, by interpreting a series of measurements (well logs). This classification is often done manually, which is very tedious and time consuming. Automatizing such work-flow is usually done by applying *machine learning* algorithms, and it could help the interpreters to understand better the patterns between measurements.

Machine learning is very popular in the moment, and highly used by *Statisticians* and *Data Scientists* to help the analysis of *big data*. Machine learning is a sub-area of the *Artificial Intelligence* field, and it consists mostly on doing predictions by recognizing patterns on the data *features* by using statistical learning methods (Hastie et al., 2001). Recently, the geoscientists are applying machine learning algorithms combining with different processing and interpretation methods. Maybe the most common application is for facies classification, by the use of ensemble classifiers (Bestagini et al., 2017; Zhang and Zhan, 2017; Caté et al., 2017), neural networks, or NN (Silva et al., 2014), and support-vector machines, or SVM (Caté et al., 2017; Alexsandro et al., 2017; Wrona et al., 2018), but machine learning also has other applications in geophysics, such as in FWI, by using convolutional neural, or CNN, networks for salt identification (Lewis and Vigh, 2017), and or using FLEXWIN for time-window selection (Chen et al., 2017). It is possible to find works on trace interpolation using support-vector regression, or SVR (Jia and Ma, 2017), or by using Monte-Carlo approximations (Jia et al., 2018). Deep neural networks (DNN) is used by Araya-Polo et al. (2017) for fault detection and by Araya-Polo et al. (2018) for

*CREWES - University of Calgary

tomography. Nearest neighbors (k-NN) can be implemented to help on the CMP velocity analysis (Smith, 2017). Russell et al. (2002) combine NN with AVO. Many others machine learning algorithms applications can be found in the literature.

Hall (2016) proposed a contest to use machine learning for facies classification, and the results of the contest were published on Hall and Hall (2017). Caté et al. (2017) shows that ensemble classifiers, such as *Random Forest* and *Gradient Boosting Trees*, are a more suitable choice to identify the rocks types using well logs. This paper is focused to classify Hall (2016)'s well log data by pre-processing the logs data, applying data imputation, and classifying the rocks using the package XGBoost for Gradient Boosting Trees (Chen and Guestrin, 2016). Programing language R (R Core Team, 2018) and RStudio (RStudio Team, 2015) are used for the analysis and the resulting *RMarkdown* is attached in the *Appendix*, and shows that accuracy of the predictions are higher (an increase of 10 percentage points) if the data is pre-processed (cleaned, data imputation, and feature engineering).

TREE-BASED METHODS

In this paper, the model used for predictions is the *Gradient Boosting Tree* (Hastie et al., 2001), which is a guided ensemble of *Decision Trees*. In this section, the discussion starts with the definition of *regression* and *decision trees*, introducing the Gradient Boosting Tree lately. The theoretical development in this paper follows the one showed by Hastie et al. (2001) and the algorithm called *CART* (Classification and Regression Trees) is the one used in the research.

Background

Short explanation, tree-based methods split the feature (variable) space in rectangles and assign a simple model (like a constant) in each one, as shown in figure 1 (each component will be explained soon). It is conceptually simple, but shows to be powerful.

First, let's solve the regression problem. Consider we have a continuous response Y and input features (variables) X_1 and X_2. The idea is to create a set of partitions R_m in the (X_1, X_2) space (figure 1a and b) and "predict" a constant c for each partition. Assuming binary partitions, the partitions of figure 1a is unlikely to happen. Figure 1b shows partitions more realistic to the algorithm. The formulation of such partitions is:

$$\hat{f}(X) = \sum_{m=1}^{M} c_m I\{(X_1, X_2) \in R_m\} \tag{1}$$

where $\hat{f}(X)$ are the predicted constants c_m response for the feature space $X = (X_1, X_2)$ in the partition R_m. $I = 1$ while the features (X_1, X_2) are inside of each one of the M partitions R_m. The partitions are included in the model each at a time, by selecting the t values, as the tree of figure 1c grows. Figure 1d is the resultant regression of equation 1, where the vertical axis represents $\hat{f}$. This kind of method (tree) has the advantage of interpretability, as it is build over a series of *if* conditions.

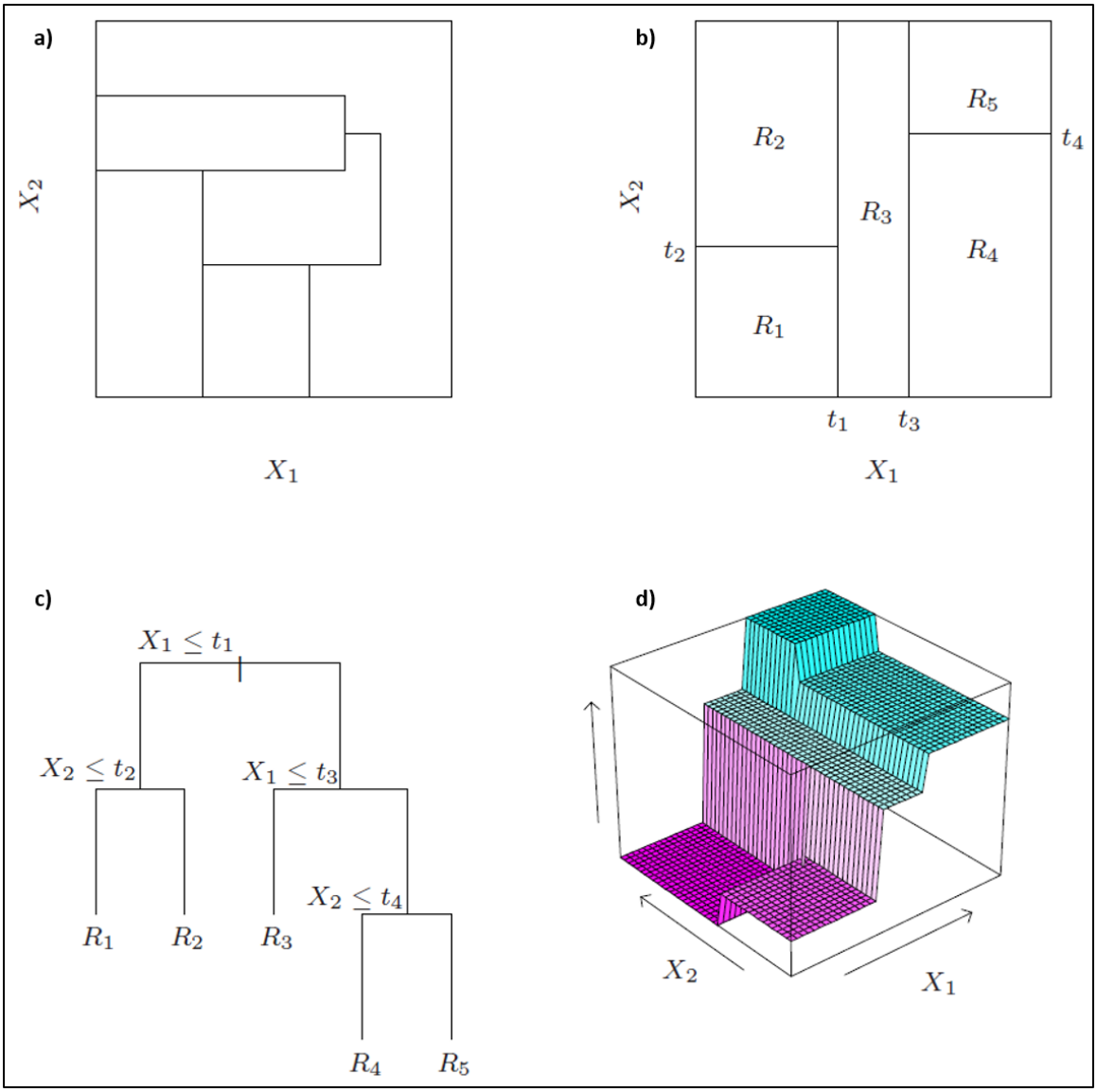


FIG. 1. Diagram of a tree. a) shows a set of partitions not obtainable from recursive binary splitting. b) shows a set of boundaries obtained on 2D feature space by recursive binary splitting. c) is the tree obtained, corresponding panel a). d) is the predicted surface obtained by the tree. Figure modified from Hastie et al. (2001).

The next step is to start to describe how to grow a tree, starting by a regression tree.

Regression trees

Let's assume now that our data consists of p inputs (features) and one response for each of the N observations. In others words, the data is represented by (x_i, y_i) for $i = 1, 2, \ldots, N$, where N is the number of rows, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$, where p is the number of columns minus one. Now, consider M partitions $(R_1, R, 2, \ldots, R_M)$, the equation 1 is:

$$f(x) = \sum_{m=1}^{M} c_m I\{x \in R_m\} \tag{2}$$

The goal of the algorithm is to automatically decide which feature and points to split (the feature to use the *if* condition, and with which value or rule), so the solution becomes something similar to the figure 1c. For that, equation 2 needs to be optimized. A way for that is to minimize the *residual sum of squares*, or *RSS* (in geoscience, it is called *objective function*. In data science and statistics, it is called *cost function*):

$$Q(f) = \sum_{i=1}^{N} [y_i - f(x_i)]^2 \tag{3}$$

The minimization of equation 3 leads to optimized $\hat{c}_m$ that is just the average of y_i in region R_m:

$$\hat{c}_m = ave(y_i | x_i \in R_m) \tag{4}$$

Hastie et al. (2001) suggest the minimization of the RSS to find the best partition is not feasible in computational terms, so a *greedy algorithm* is usually chosen. The procedure is to set a splitting point s and splitting variable (feature) j, and the partitions are:

$$R_1(j,s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j,s) = \{X | X_j > s\} \tag{5}$$

Then find the values of j and s that solves the following equation:

$$\min_{j,s} \left[\min_{c_1} \sum_{x_1 \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_1 \in R_2(j,s)} (y_i - c_2)^2 \right] \tag{6}$$

For each choice of j and s, the values of c_1 and c_2 of the inner minimization are solved by:

$$\hat{c_1} = ave(y_i | x_i \in R_1(j,s)) \quad \text{and} \quad \hat{c_2} = ave(y_i | x_i \in R_2(j,s)) \tag{7}$$

In others words, the choice of feature and point splitting parameters are done by trying them all (or a selected batch of them) and select the ones that minimize equation 6.

Train and test errors

The next step is to decide how deep (long) the tree must be. It is intuitive to imagine that a too large tree will solve all the points of the data. But will this model (tree) be good for a different dataset? Probably not, as the model is *overfitting* the first dataset. If the goal of the created model is to do predictions on another dataset, we need to introduce the idea of *train and test errors*.

Given a dataset D that consists of p features and one response for N observations, we can split it into two datasets: the training D_{train} dataset with N_{train} observations and test D_{test} dataset with N_{test} observations. D_{train} and D_{test} have the same number of features and response as the data D, and $N = N_{train} + N_{test}$.

The model is created (or trained) using the training set D_{train} by minimizing equation 3, the *train error*. Figure 2 illustrates the behavior of the train error as the tree gets larger (increasing the model complexity), represented by the solid blue curve. With a model sufficient complex, the error goes to zero. The *test error* is computed using equation 3 by applying the trained model on test data D_{test}, and its behavior over model complexity is represented by the solid red curve in figure 2. It reaches a minimum point on a certain model complexity, then the error starts to increase, while the train error keeps decreasing. That is the point when the model is overfitting the train data.

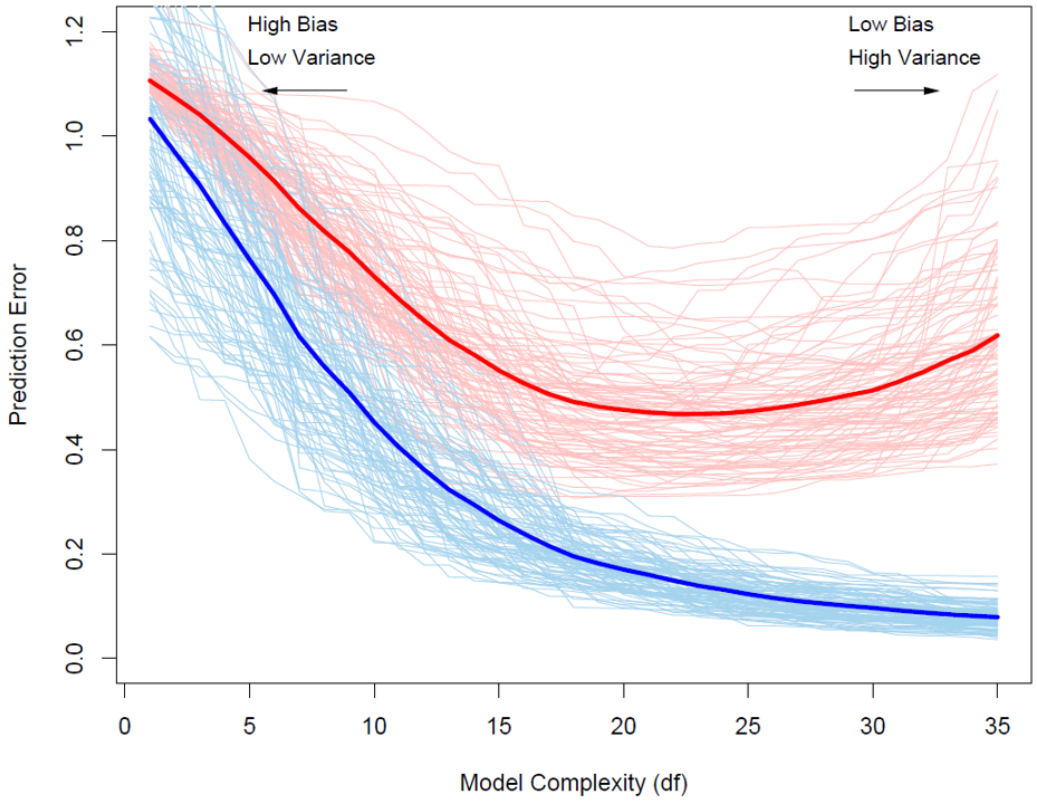


FIG. 2. The plots of train (light blue) and test (light red) errors. The curves were calculated using 100 different datasets with size of 50 each one. The solid curves are the expected errors (the average of all the curves). Figure from Hastie et al. (2001).

The "best" model is the one that minimizes the test error. It can be obtained by training the model over the train data and using the test data to evaluate the model, to reach the optimal balance between the errors.

Another way could be the implementation of a *tuning parameter* Hastie et al. (2001), in order to penalize large models. First grow a large tree T_0, then define a subtree $T \subset T_0$ that is obtained by pruning (collapsing internal nodes) T_0. The terminal (last) nodes receive the index m, and the node m is related to the partition R_m. Let's denote the number of terminal nodes in T by $|T|$. Using:

$$\begin{aligned} \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2 \end{aligned} \tag{8}$$

Now we can define the cost complexity criteria:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T| \tag{9}$$

where α is the *tuning parameter* and it governs the tradeoff between the tree size and its precision to fit the data. A large α result in smaller trees, as small α can lead to overfitting. This parameter must be chosen carefully.

Decision (classification) trees

At this point the idea of regression tress has been presented. But the goal of this paper is to *classify* outcomes that can take values of $1, 2, \ldots, K$. *Decision tree* is a classification algorithm and it is similar to the regression trees, with some changes on the criteria for splitting nodes and pruning the tree. Now, the algorithm will split and classify an outcome based on the majority class in the current depth of the tree (node). In a given node m on region R_m with N_m observations, the proportion of class k counted in the node is:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{10}$$

The classified observation in node m will be the class $k(m) = \arg\max_k \hat{p}_{mk}$. We can compute the *misclassification error* by:

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq k) = 1 - \hat{p}_{mk} \tag{11}$$

The algorithm can keep splitting and growing the tree in order to minimize equation 11, but the user needs always to be aware of the train and test errors, and must select the best parameters for a good balance.

Boosting trees

Boosting has the premise to combine many "weak" classifiers (the ones such accuracy is slightly higher than random guess) to produce a strong classifier. Boosting is one of the most powerful learning methods. It was originally designed for classification, but it has recently been expanded to regression as well.

Classification and regression trees were discussed previously. Just as a reminder, both methods divide the feature space into partitions $R_j, j = 1, 2, \ldots, J$, and each partition is assigned to a constant (or classifier) γ_j:

$$x \in R_j \Rightarrow f(x) = \gamma_j$$

And the tree is expressed as:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{12}$$

where the parameter $\Theta = \{R_j, \gamma_j\}_1^J$ and are found by minimizing

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j), \tag{13}$$

in which $L(y_i, \gamma_j)$ is called the *loss function*, and it can be equal the cost function of equation 3 for regression, or the misclassification error of equation 11 for classification problems. Now the optimization problem is to find γ_j and R_j. Well, as showed before, for a given R_j, $\gamma_j = \bar{y}_j$ (average of $y_j, x \in R_j$). The difficult part is to find R_j. Usually approximate solutions are used, such as a greedy, top-down recursive partitioning algorithm. In general, equation 13 is approximated by a smoother criterion to optimize R_j:

$$\tilde{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i, \Theta)), \tag{14}$$

The boosted tree is a combination of all the "weak" trees, and the output is an optimized tree. Hastie et al. (2001) point the final tree as the sum of all the trees:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{15}$$

Equation 15 is induced in a *forward stagewise manner*. This method approximates the solution of equation 14 by adding new functions (in this case a new tree) to the model without changing the parameters. Consider the squared-error loss:

$$L(y, f(x)) = [y - f(x)]^2 \tag{16}$$

For each iteration m, the forward stagewise manner solves for an optimal function $b(x; \gamma_m)$ and a corresponding coefficient β_m, to be added to the current function (or tree) $f_{m-1}(x)$, producing the new function $f_m(x)$, and then the repeat process again. In the end, this is an iterative method. So, equation 16 becomes:

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= [y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma)]^2 \\ &= [r_{im} - \beta b(x_i; \gamma)]^2 \end{aligned} \tag{17}$$

where $r_{im} = [y_i - f_{m-1}(x_i)$ is the residual of the current model on the ith iteration. This optimization is better expressed on algorithm 1.

Algorithm 1: Forward stagewise additive modeling

Initialize $f_0(x) = 0$

for *m = 1 to M* **do**

Compute: $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$

Set: $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$

Now, with the forward stagewise manner method introduced, the parameters of equation 14 are iteratively solved:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta_m)) \tag{18}$$

for regions and constants $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{J_m}$ of the next tree (model), given the current model $f_{m-1}(x)$. As the region R_{jm} is set, the optimal values for the constants γ_{jm} are computed:

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{19}$$

Equation 18 can be numerically optimized with any differentiable loss function. The loss in setting $f(x)$ to predict y is given by:

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{20}$$

We want to minimize the loss $L(f)$ with respect to f, where $f(x)$ is computed from equation 15. Now let's assume a *generic function* $f(x)$ (not necessarily a tree). The minimization can be expressed as a numerical optimization:

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}) \tag{21}$$

where $\hat{\mathbf{f}} \in \mathbb{R}^N$ are the approximations of function $f(x_i) \rightarrow \hat{\mathbf{f}}\{f(x_1), f(x_2), \ldots, f(x_N)\}$. The numerical optimization will assume the form:

$$\hat{\mathbf{f}}_M = \sum_{m=0}^{M} \hat{\mathbf{h}}_m, \quad \hat{\mathbf{h}}_m \in \mathbb{R}^N \tag{22}$$

where $\hat{\mathbf{h}}_m$ is a step function, that is solve iteratively (such as the trees for the forward stagewise manner). Following, we will show that the step function can be replaced by the *gradient boosting trees*, a *steepest descent* solution given a loss function.

Steepest descent

The steepest descent method replaces the step function $\hat{\mathbf{h}}_m$ of equation 22 for $-\rho_m \mathbf{g}_m$, where ρ_m is a scalar, named *step length*, and $\mathbf{g}_m \in \mathbb{R}^N$ is the partial derivative (gradient) of the loss function $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$. The components of the gradient are:

$$g_{im} = \left[\frac{\partial L(x_i, f(x_i))}{\partial f(x_i)} \right] \tag{23}$$

And the step length is:

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1-\rho \mathbf{g}_m}) \tag{24}$$

Where the current solution is then updated:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m \tag{25}$$

This process is then repeated at the iteration. Steepest descent is considered as a greedy solution, as the computed gradient is the local direction that most rapidly decreases the loss function at the current iteration. The most commonly used loss functions for minimization and their corresponding gradient are show on table 1.

Table 1. Gradients for the most commonly used loss functions.

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	sign$[y_i - f(x_i)]$
Classification	Deviance	kth component: $I(y_i = G_k) - p_k(x_i)$

Gradient boosting trees

The gradient boosting is the solution of the steepest descent when a loss function is given and the gradient can be mathematically and numerically computed. The algorithm 2 represents a generic gradient boosting tree algorithm for regression.

Algorithm 2: Gradient boosting tree

Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$

for *m = 1 to M* **do**

  for *i = 1 to N* **do**

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]$$

  Fit a tree to the targets r_{im} setting terminal regions $R_{jm} = 1, 2, \ldots, J_M$

  for *j = 1 to* J_m **do**

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma)$$

  Update:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

Output: $\hat{f}(x) = f_M(x)$

Note that, differently from forward stagewise manner, the parameters, at each iteration, are computed by fitting a tree to the gradient:

$$\tilde{\Theta}_m = \arg\min_\Theta \sum_{i=1}^{N} [-g_{im} - T(x_i; \Theta)]^2 \tag{26}$$

The gradient is the partial derivative of the loss function with respect to the tree. Table 1 shows the gradient for the most commonly used loss functions. The $L1$ and $L2$ norms are usually used for regression.

For classification the *deviance* is usually chosen. For a problem with K classes, the response Y takes values of the unordered set $\zeta = \{\zeta_1, \zeta_2, \ldots, \zeta_K\}$ that are the classes. We need to find a classifier $G(x)$ that takes the values of ζ. For a given point x, we can compute the probability of the class classification $p_k(x) = \Pr(Y = \zeta_k|x)$, $k = 1, 2, \ldots, K$, for then the Bayes classifier is:

$$G(x) = \zeta_k \quad \text{where} \quad k = \arg\max_l p_l(x) \tag{27}$$

So the classifier will select the class with the higher probability to be correct, and it can be obtained by a logistic model:

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}} \tag{28}$$

which ensures that the probabilities are between 0 and 1, with their sum equals 1. Now we can introduce a K-class *multinomial deviance* loss function:

$$\begin{aligned} L(y, p(x)) &= -\sum_{k=1}^{K} I(y = \zeta_k) \log p_k(x) \\ &= -\sum_{k=1}^{K} I(y = \zeta_k) f_k(x) + \log\left(\sum_{l=1}^{K} e^{f_l(x)}\right) \end{aligned} \tag{29}$$

For the gradient boosting tree, we can compute the gradient of equation 29 to use on algorithm 2:

$$\begin{aligned} -g_{ikm} &= \frac{\partial L(y, f_{1m}(x_i), \ldots, f_{1m}(x_i))}{\partial f_{km}(x_i)} \\ &= I(y_i = \zeta_k) - p_k(x_i) \end{aligned} \tag{30}$$

where the higher probability for the class is chosen.

Equation 26 gives us a very interesting insight about the gradient boosting method: it fits a new tree over the wrong predictions. At each iteration, the new tree will focus on the larger gradients, that are related to the larger residuals. In the classification problem, the method focus on the misclassified part of the model.

THE DATA

Hall (2016) proposed a *facies classification contest* after applying a support *vector machine algorithm* to predict the rocks classes. The data consists of a set of well logs and in-

dicators that comes from a University of Kansas class exercise on the Hugoton and Panoma gas fields.

Five well logs and two indicators are given as features of ten well locations for the analysis. The data contains a total of 4149 rows and 11 columns, the seven features, the target column (classified facies), plus some meta information, such as well name and depth. The seven features are:

1. Gamma ray (GR)
2. Resistivity (ILD_log10)
3. Photoelectric effect (PE)
4. Neutron-density porosity difference (DeltaPHI)
5. Average neutron-density porosity (PHIND)
6. Nonmarine/marine indicator (NM_M)
7. Relative position (RELPOS)

Different rock types are found in the data and they are represented (classified) by a integer value that goes from 1 to 9, as shown on table 2. It is interesting to see the column *adjacent facies*. Those are the rocks types usually found close to each one. So, this give us a priori information that misclassifications as these "neighbors" types will be more common (at least, we can expect that).

Table 2. Facies labels and their descriptions.

Facies	Description	Label	Adjacent Facies
1	Nonmarine Sandstone	SS	2
2	Nonmarine coarse siltstone	CSiS	1,3
3	Nonmarine fine siltstone	FSiS	2
4	Marine siltstone and shale	SiSh	5
5	Mudstone	MS	4,6
6	Wackestone	WS	5,7,8
7	Dolomite	DPhi	6,8
8	Packstone-grainstone	PS	6,7,9
9	Phylloid-algal bafflestone	BS	7,8

Ten wells are presented in the data. One of the wells will be separated (removed) from the dataset to be used as the blind well for the evaluation of the predictions. The well names are listed below:

1. Shrimplin
2. Alexander D
3. Shankle
4. Luke G U
5. Kimzey A
6. Cross H Cattle
7. Nolan
8. Recruit F9

9. Newby
10. Churchman Bible

Figure 3 shows the *Shankle* well and its five logs: gamma ray (red), resistivity (yellow), neutron-density porosity difference (green), average neutron-density porosity (blue), and photoelectric effect (magenta). The colors in right are the given classification of the rocks at the given depth. This well in the figure is the one that will be separated from the data and used as the blind/evaluation well.

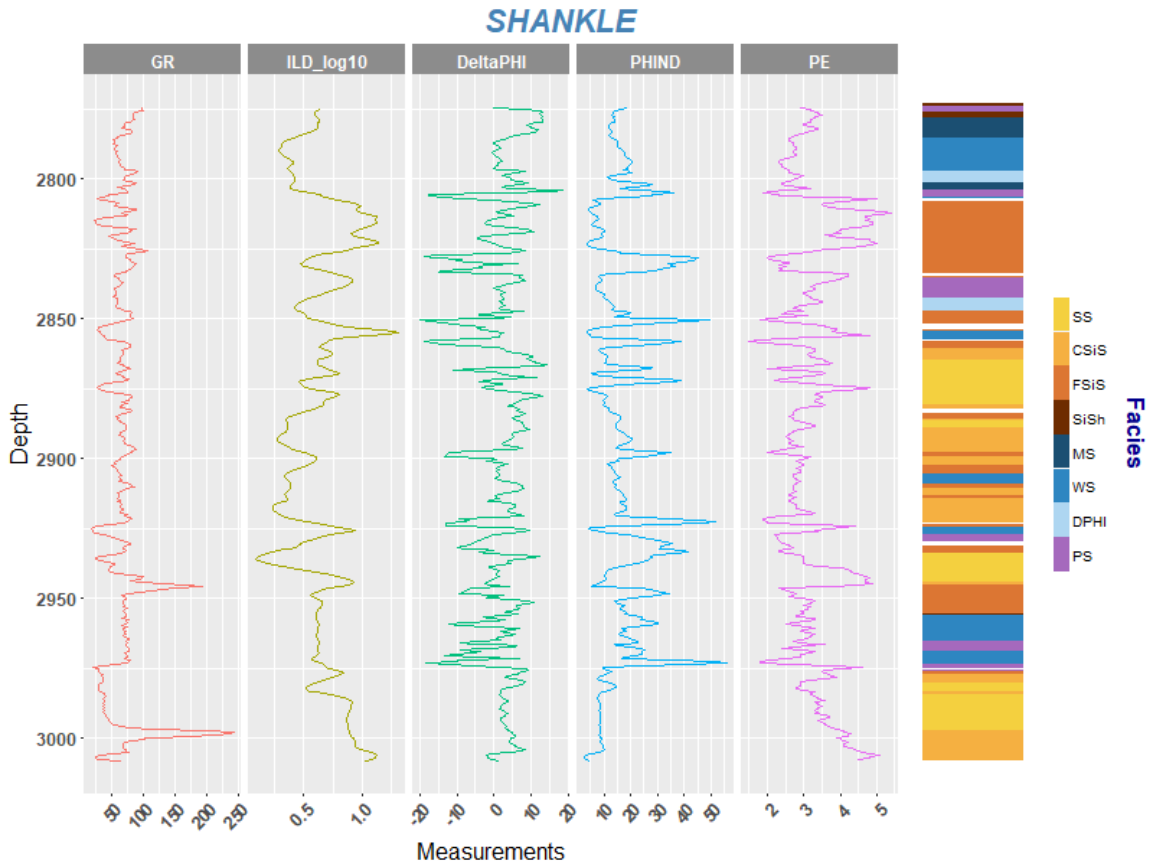


FIG. 3. The Churchman Bible well logs and the facies classification on the right.

FEATURES PROCESSING

This section is focused on the features processing, pointing the steps taken to improve the classification accuracy. It was applied data filtering (removal of "broken" data), data imputation (treatment over missing data), feature engineering (creation of new features based on the given ones), and predictions adjustments (corrections applied to the predictions).

Filtering

Initially, all the ten well logs and indicators were plotted and analyzed. The first thing that came to my eyes was the well Recruit F9 logs and facies classification (figure 4). The well has around $300m$ depth and all the logs present a weird behavior: they are very smooth with the presence of spikes. Also, it looks to be formed for only one type of rock for the

$300m$ length, the nonmarine sandstone (SS). This seems to be highly unlikely and indicates that the well may be "broken".

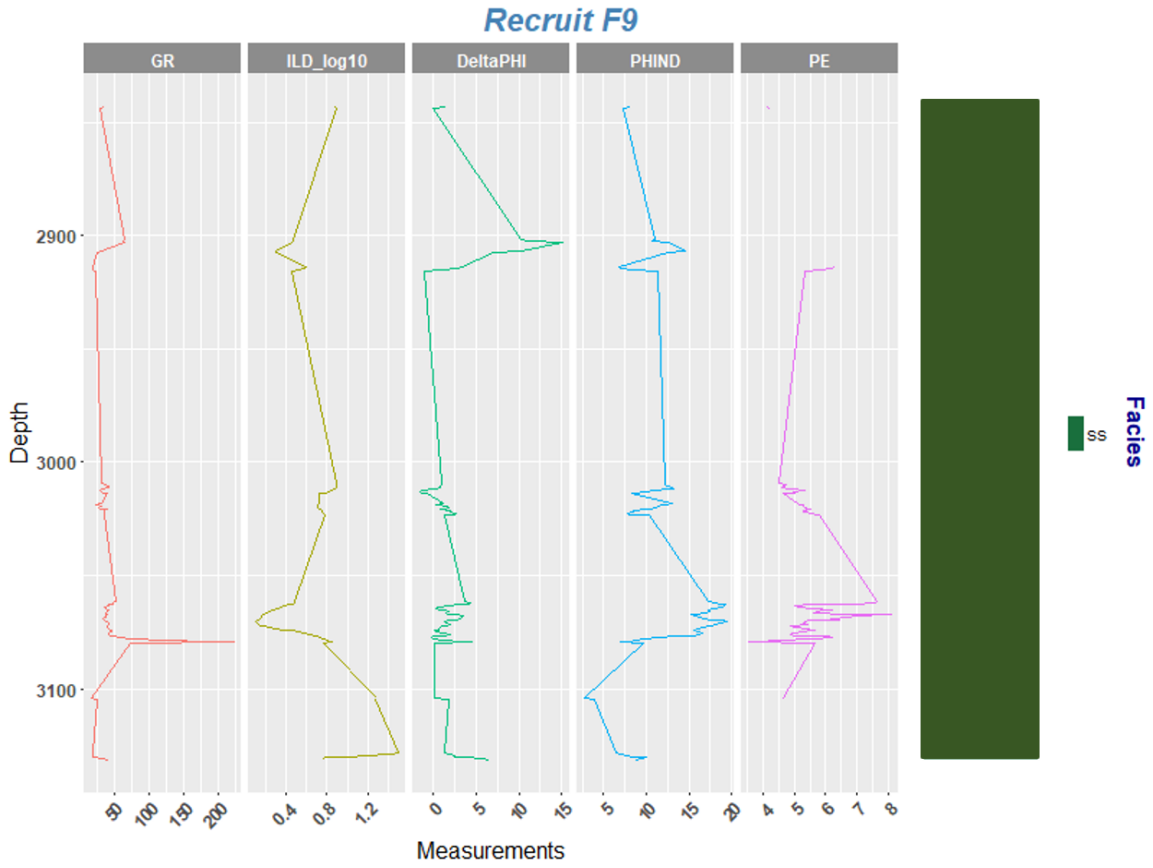


FIG. 4. Recruit F9 well shows to be "broken".

This kind of data can cause to bias the trained model (gradient boosting trees), as it shows inconsistent behavior. To avoid a biased model, this well was deleted from the data, leaving me with a total of nine wells (eight to train the model, and one for model evaluation).

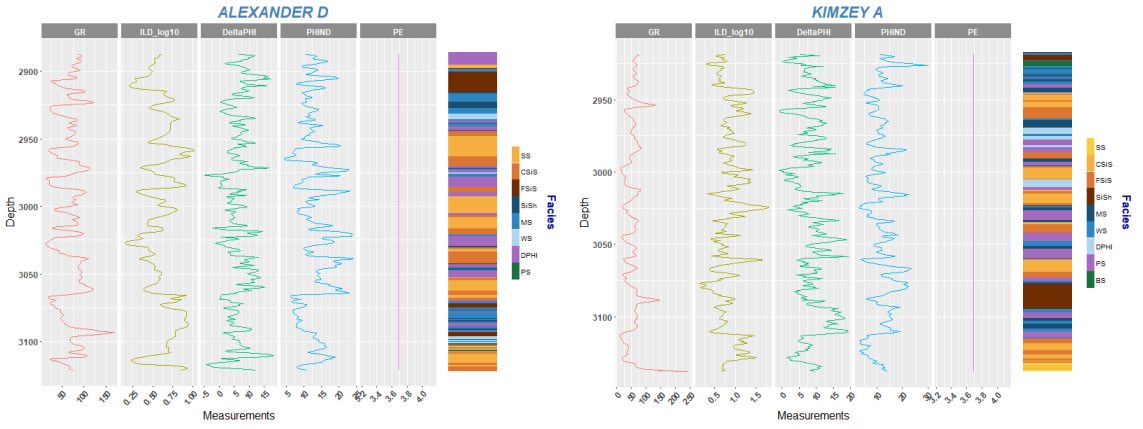
Any other filtering process was applied to the data, as it seemed unnecessary.

Data imputation

The gradient boosting tree method has the advantage to handle better missing data. But having a complete dataset usually leads to a more accurate prediction.

All the features were analyzed for missing data, and the photoelectric effect (PE) came with 905 (of 4149) missing points, or almost 22% of non-available data. Most of the solutions of the facies classification contest (Hall, 2016), such as Bestagini et al. (2017), just replace the missing data by the global mean of the feature. Well, all the missing PE data are related to two wells, the *Alexander D* (figure 5a) and *Kimzey A* (figure 5b), and the entire PE measurements are missing on both wells, as shown in figure 5. Completing the data with its global mean does not sound as an optimal solution. Maybe the data imputation requires

a more sophisticated solutions, that can lead to *predictions* of the missing PE. Gelman and Hill (2007) cite different methods for data imputation, such as random values completion, nearest neighbors, regression, and others.



(a) (b)

FIG. 5. Missing data on the wells (a) Alexander D, and (b) Kimzey A. The photoelectric effect is completely missing. For plotting purposes, the missing PE was filled with the global mean.

The data contains 7 complete wells and 2 with the missing data. As all the logs are giving a measurement of each rock type, I believe the logs may contain some relationship (they are not independent variables). A regression model can be created by training it on the 7 good wells and then predict the PE values on the other 2 wells. The PE values can be "recovered" from the other logs. Equation 31 is a multivariate regression model for the PE values:

$$PE = \alpha_0 + w_1 * GR + w_2 * ILD\_log10 + w_3 * DeltaPHI + w_4 * PHIND \quad (31)$$

where α_0 is the intercept and the w's are the slopes for each feature.

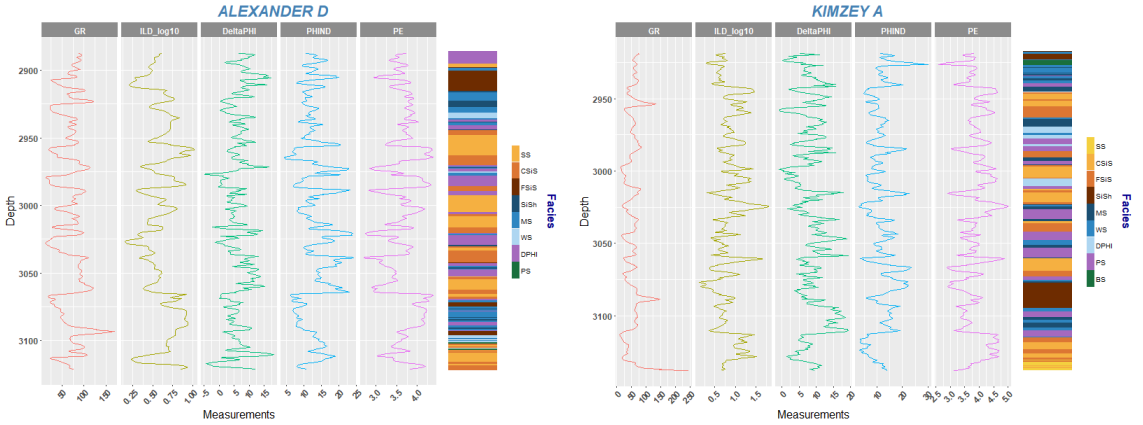
By training the regression model of equation 31 with the 7 good wells, all the coefficients were computed, as shown on equation 32, and the results are plotted in figure 6.

$$\begin{aligned} PE = 4.394027 - 0.00284 * GR + 0.489608 * ILD\_log10 + \ldots \\ - 0.01147 * DeltaPHI - 0.05867 * PHIND \end{aligned} \quad (32)$$

The coefficients of equation 32 point feature PHIND as the one giving the largest contribution on predicting PE.

Feature engineering

Bestagini et al. (2017) proposes the use of *feature augment*, or *feature engineering*, that consists of creating new features from the available ones. Making different plots of



(a) (b)

FIG. 6. Predicted photoelectric effect using multivariate regression for (a) Alexander D, and (b) Kimzey A.

the data can help to identify patters on the data, and the feature engineering can be done wisely. Figure 7 is the pairs plot of all the well logs (indicators are not included). The colors of the points are the rocks classifications. The diagonal are the histogram plot with the classification counts. The bottom left are the scattering plots, and the top right are the density plots.

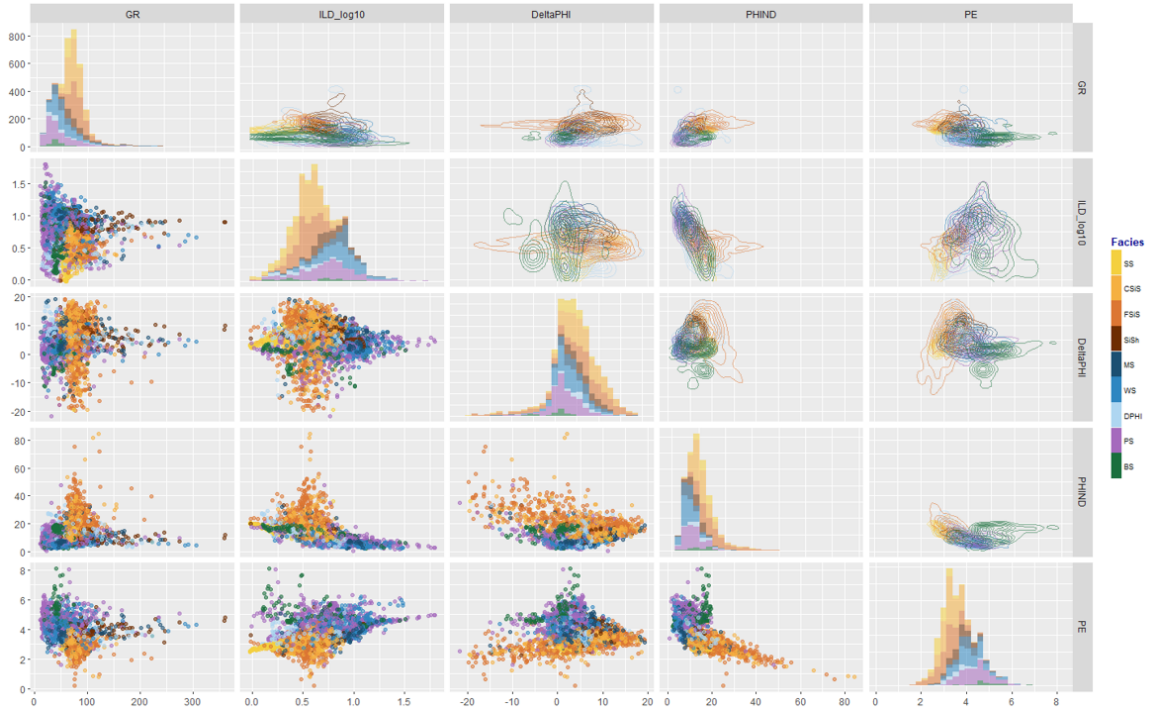


FIG. 7. Pairs plot (or scattering plots matrix), of all the well logs. The colors represent facies classification. Note at the circular behavior of the features. The diagonal are the histogram plot with the classification counts. The bottom left are the scattering plots, and the top right are the density plots.

The most notorious observation taken from figure 7 is the circular behavior of the plots. When creating a classifier model, it is complicated to define circular classification bound-

aries. One way to deal with it is to transform the features to a domain where circular patterns become more linear. In the case of the well logs, a *polar coordinates* transformation can do the trick.

Assume we have 2 continuous features X_1 and X_2. The polar coordinates transformation will return 2 new features for each pair of feature, the the radial coordinate r and the angle ϕ, and the conversion follow the relations of equation 33:

$$\begin{aligned} r &= \sqrt{{X_1}^2 + {X_2}^2} \\ \phi &= \arctan\left(\frac{X_2}{X_1}\right) \end{aligned} \tag{33}$$

Those sets of relations were applied in the data combining all the pairs of well logs (the indicators were not used), and from 7 initial features, I ended up with 27.

Another feature engineering used in this work is proposed by Bestagini et al. (2017), where the features gradients are computed, according to equation

$$\bigtriangledown_{f_i}^d = \frac{\mathbf{f}_i^{d-1} - \mathbf{f}_i^d}{\Delta Depth} \tag{34}$$

where $\bigtriangledown_{f_i}^d$ is the gradient of the feature $\mathbf{f}_i$ at depth d. The gradient is computed over the features *after* the polar coordinates conversion, and the final number of features are raised to 54.

Clustering: k-means

Clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). In other words, in a dataset with N observations, can be assigned to a cluster $k \in \{1, \ldots, K\}$, with $K < N$, and K is the total number of clusters. These assignments can be characterized by an *encoder* $k = C(i)$, that assigns the ith observation to the kth cluster, based on the dissimilarities (or distance) $d(x_i, x_{i'})$, based on the observations x.

The goal is to assign close points to a cluster, so the minimization can be applied to a loss function of the form:

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k} d(x_i, x_{i'}) \tag{35}$$

A very popular algorithm for clustering analysis is the *k-means* (Lloyd, 1982), which is applied to quantitative type data, and uses the Euclidean distance:

Algorithm 3: K-means clustering

1 - For a given cluster assignment C, the total cluster variance, obtained from equation 41, is minimized to $\{m_1, \ldots, m_k\}$ yielding the means of the currently assigned clusters (equation 40).

2 - Given a current set of means $\{m_1, \ldots, m_k\}$, equation 41 is minimized by assign each observation to the closest (current) cluster mean (center):

$$C(i) = \arg \min_{1 \leq k \leq K} \|x_i - m_k\|^2 \tag{36}$$

3 - Steps 1 and 2 are repeated until the assignments do not change.

$$d(x_i, x_{i'}) = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = \|x_i - x_{i'}\|^2 \tag{37}$$

Equation 35 can be written as:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \|x_i - x_{i'}\|^2 = \sum_{k=1}^{K} N_k \sum_{C(i')=k} \|x_i - \bar{x}_k\|^2 \tag{38}$$

where $\bar{x} = (\bar{x}_{1k}, \ldots, \bar{x}_{pk})$ is the mean vector associated with the kth cluster, and $N_k = \sum_{i=1}^{N} I(C(i) = k)$. The minimization criteria is to assign all the N observations to all the K clusters in a such way that $\bar{x}_k$ is minimized. This can be obtained by an iterative algorithms that solves:

$$C^* = \min_{C} \sum_{k=1}^{K} N_k \sum_{C(i')=k} \|x_i - \bar{x}_k\|^2 \tag{39}$$

by noting that for any set of observations S, for the cluster with mean (center) m:

$$\bar{x}_S = \arg \min_{m} \sum_{i \in S} \|x_i - m\|^2 \tag{40}$$

C^* can be obtained by solving:

$$\min_{C, \{m_k\}_1^K} \sum_{k=1}^{K} N_k \sum_{C(i')=k} \|x_i - m_k\|^2 \tag{41}$$

This minimization is done by an alternating optimization procedure given in algorithm 3, which is used, in this analysis, to create a new feature *cluster*. It inputs all the original and engineered features to the algorithm and estimate the closest observations into the chosen number of clusters K. It is expected that this new feature helps to improve the gradient boosting predictions.

Predictions adjustments

To avoid overfitting, the gradient boosting model is created to be relatively smooth, meaning that, if we have a segment of classified rocks that are most sandstone, for example, it would be unlikely that inside this segment a single mudstone is correctly classified. As the rock types are categorized as numbers (1 to 9), a *median filter* with fixed window length can be applied to remove single classifications. I use a window with fixed length of 5 depth steps.

FACIES CLASSIFICATION

First predictions

Train and test data are split by selecting *Shankle* as the blind/evaluation well. *Recruit F9* is deleted from the data, as mentioned previously, due to its strange behavior and my lack of trust on it. And that is all the process done for the first predictions.

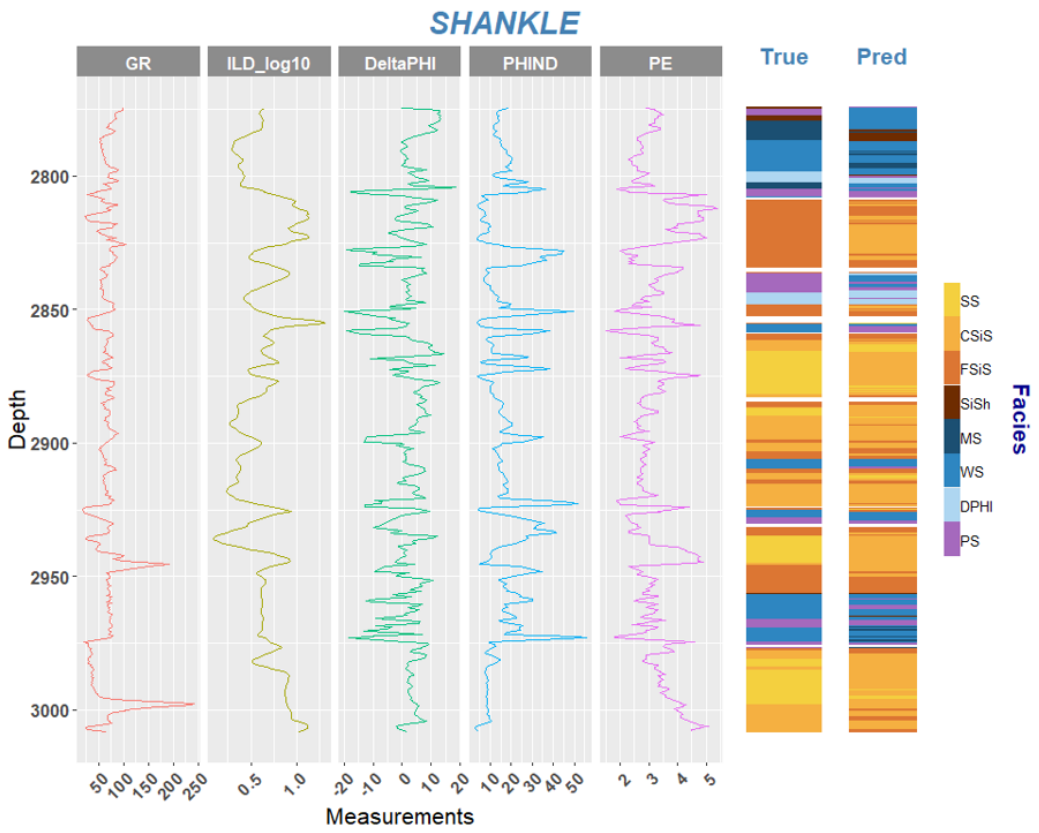


FIG. 8. Predictions on the blind/evaluate well without data processing.

A gradient boosting tree model were trained on the 8 train wells with 7 features each, and the optimum model after 30 iterations, with an accuracy of 47%. Figure 8 compares the true and predicted facies classification for this firs test.

The predictions worked reasonably well, finding properly most of the beginning and ending of the large segments, and several misclassifications in between interfaces. The shallow part of the well showed to be tricky and is mostly misclassified. The next step is to try improving the predictions.

Tuned predictions

In the second part of the tests, all the processing is applied on the train and test datasets (the processing is not done at the target column). Now, instead of 7 features to run the gradient boosting model, 55 features are available (including the clustering process). For the k-means algorithm, the observed data is assigned to 8 different clusters. The expectations are that with more features, the model can recognize different patterns and create more optimized classification boundaries.

Model training was completed after 27 iterations, using the same 8 wells as before, but including the new features, and the new predictions had an improved accuracy of 60%. Figure 9 shows the new predictions compared to the true classification.

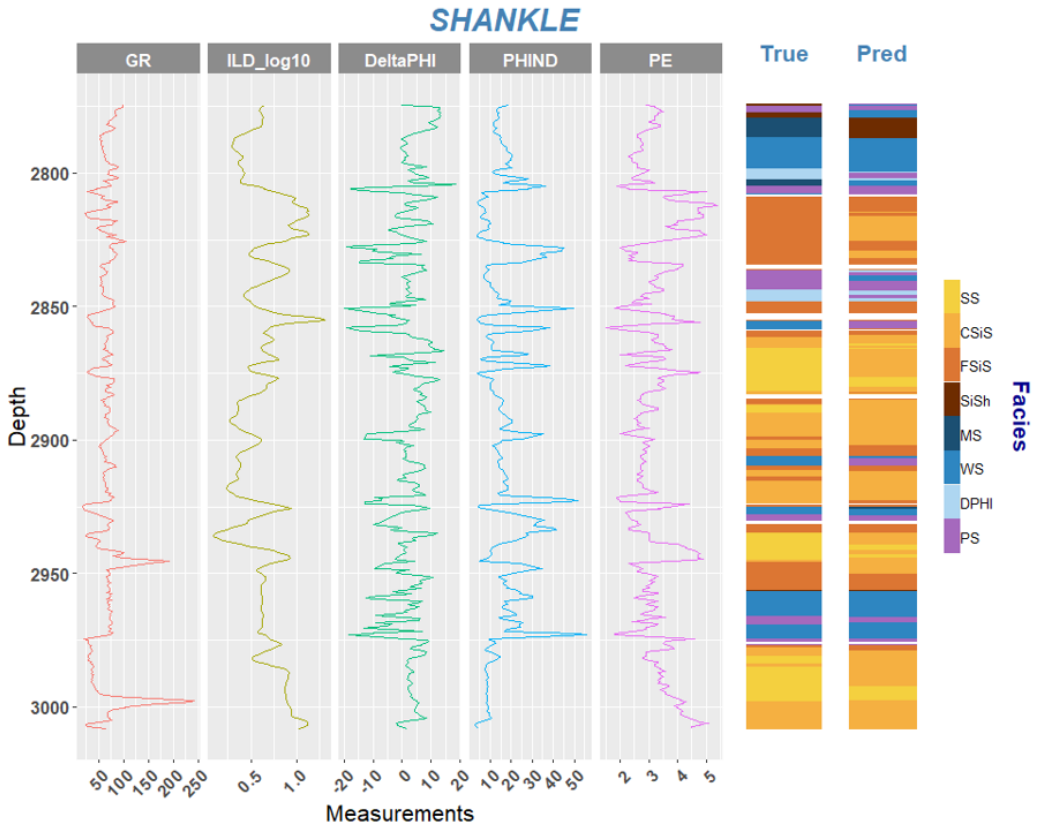
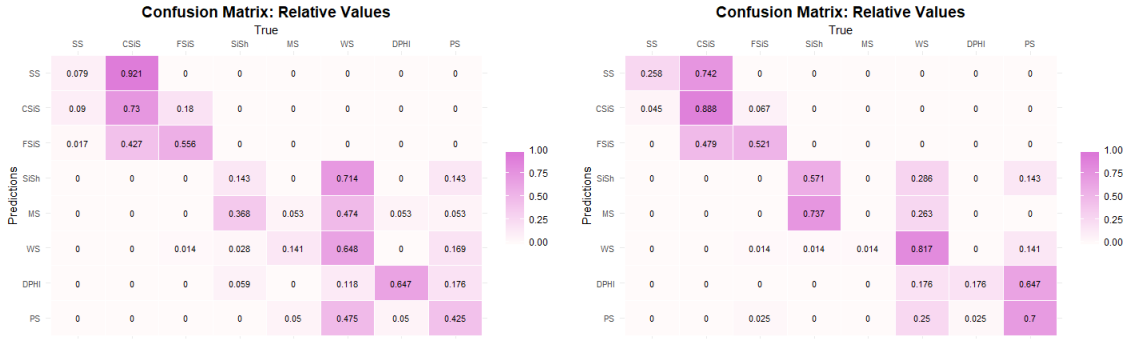


FIG. 9. Tuned predictions on the blind/evaluate well after data processing.

The new predictions are visibly more consistent to the true classification, but the shal-

low part is still tricky. Predictions now are more smooth due to the application of median filter, but improved the overall accuracy. Without the filter, predictions accuracy is 55%. The improvement of the final predictions is better demonstrated on the confusion matrices of figure 10.



(a) (b)

FIG. 10. Confusion matrix of the prediction on the blind well with (a) raw data and (b) processed data.

Figures 10a and 10b show the proportion of the rocks that are classified as the correct one, or any of the misclassified ones. The expectation, for a perfect classifier, that the diagonal of the confusion matrix is populated by 1's. From the initial to final predictions, most of the misclassifications started to focus closer to the diagonal, showing the data processing was effective. Most, but not all, of the rocks are better classified (the big exception is the dolomite). Figure 10b also shows that the misclassification of each rock tends to be at the adjacent facies (table 2), in accordance with the initial hypothesis.

The most important features for the facies classification can also be determined by their coefficients values. The highest ones are the most important. On figure 11 the 10 most important are listed, with their relative importance to the first place.

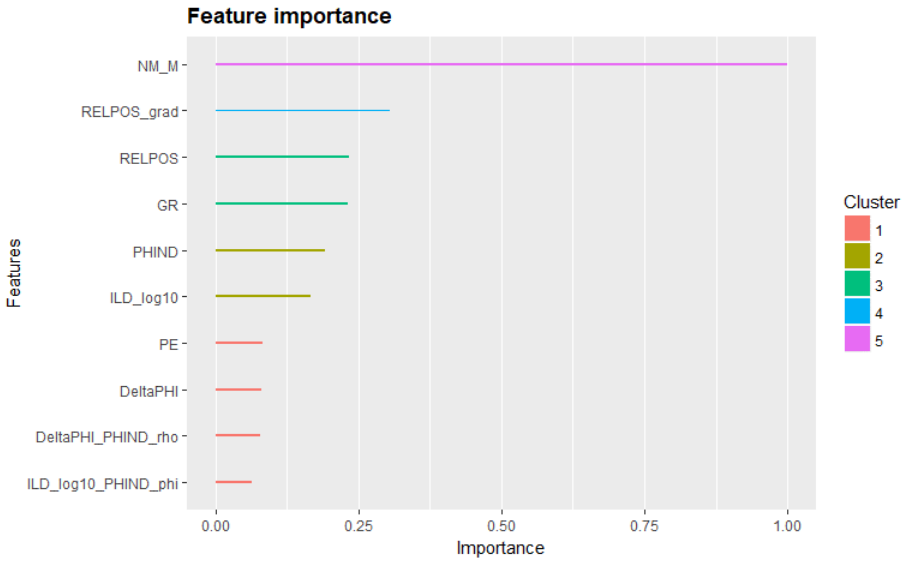


FIG. 11. The 10 most important features for the facies classification.

Nonmarine/marine indicator (NM_M) is presented as the most important feature. It is a binary vector that assumes values 1 or -1, and it should give "half" of the answer. So, it makes sense to be on first place. The following ones are the gradient of the relative position and the relative position itself. The next 5 places are the original features, followed by the polar coordinates transformed ones. The feature engineering improved the predictions with relative high importance. This shows how important it is to recognize patterns of the data before the training step, helping to train a more precise model.

CONCLUSIONS

Gradient boosting trees were used for facies classification using the dataset from a 2016 contest. The data needed to be analyzed and filtered. It was showed that the processed data increased the predictions accuracy in 13 percentual points, from 47% to 60%.

It was verified that the trained model can handle missing data, and is less susceptible to overfitting choosing the right parameters. However, predictions accuracy is improved by processing the data. First, PE missing data were "recovered" using a regressing model and a relationship between the others well logs.

Feature engineering improved was applied in two steps. For the first one, analyzing the pairs plots of the well logs and the rocks classifications, circular patterns were observed and the polar coordinate transformation was used to create new features. The second step was to calculate the gradient over depth of each feature. The final step was to use all the available features to assign similar observations into clusters.

The predictions were adjusted by a median filter, and the filtered classification showed to be smoother and more coherent to the true classification of the blind well. The confusion matrix pointed that the predictions converged better to the true values after the data processing.

In the end, machine learning algorithms can help the geoscientists on the daily job by point new insights and/or automatizing tedious work.

ACKNOWLEDGMENTS

The authors thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors and NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13, and the financial support from Canada First Research Excellence Fund. I thank Soane Mota dos Santos and Erick Gomes Anastacio for the suggestions, tips and productive discussions.

REFERENCES

Alexsandro, G. C., da P. Carlos, A. C., and Geraldo, G. N., 2017, Facies classification in well logs of the Namorado oilfield using Support Vector Machine algorithm, 1853–1858.

Araya-Polo, M., Dahlke, T., Frogner, C., Zhang, C., Poggio, T., and Hohl, D., 2017, Automated fault detection without seismic processing: The Leading Edge, **36**, No. 3, 208–214.

Araya-Polo, M., Jennings, J., Adler, A., and Dahlke, T., 2018, Deep-learning tomography: The Leading

Edge, **37**, No. 1, 58–66.

Bestagini, P., Lipari, V., and Tubaro, S., 2017, A machine learning approach to facies classification using well logs, 2137–2142.

Caté, A., Perozzi, L., Gloaguen, E., and Blouin, M., 2017, Machine learning as a tool for geologists: The Leading Edge, **36**, No. 3, 215–219.

Chen, T., and Guestrin, C., 2016, Xgboost: a scalable tree boosting system: 22nd SIGKDD Conference on Knowledge Discovery and Data Mining.

Chen, Y., Hill, J., Lei, W., Lefebvre, M., Tromp, J., Bozdag, E., and Komatitsch, D., 2017, Automated time-window selection based on machine learning for full-waveform inversion, 1604–1609.

Crampin, T., 2008, Well log facies classification for improved regional exploration: Exploration Geophysics, **39**, No. 2, 115–123.

Gelman, A., and Hill, J., 2007, Data Analysis using Regression and Multilevel/Hierarchical Models, Analytical methods for social research: Cambridge University Press, United Kingdom.

Hall, B., 2016, Facies classification using machine learning: The Leading Edge, **35**, No. 10, 906–909.

Hall, M., and Hall, B., 2017, Distributed collaborative prediction: Results of the machine learning contest: The Leading Edge, **36**, No. 3, 267–269.

Hastie, T., Tibshirani, R., and Friedman, J., 2001, The elements of statistical learning - data mining, inference, and prediction: Springer, second edn.

Jia, Y., and Ma, J., 2017, What can machine learning do for seismic data processing? an interpolation application: GEOPHYSICS, **82**, No. 3, V163–V177.

Jia, Y., Yu, S., and Ma, J., 2018, Intelligent interpolation by monte carlo machine learning: GEOPHYSICS, **83**, No. 2, V83–V97.

Lewis, W., and Vigh, D., 2017, Deep learning prior models from seismic images for full-waveform inversion, 1512–1517.

Lloyd, S. P., 1982, Least squares quantization in pcm: IEEE Transactions on Information Theory, **28**, 129–137.

R Core Team, 2018, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria.
URL `https://www.R-project.org/`

Richard F. Wadleigh, J., and Ward, J. A., 1984, Carbonate-anhydrite facies determination in the Paradox Basin by quantitative seismic stratigraphy, 497–500.

RStudio Team, 2015, RStudio: Integrated Development Environment for R, RStudio, Inc., Boston, MA.
URL `http://www.rstudio.com/`

Russell, B., Ross, C., and Lines, L., 2002, Neural networks and avo: The Leading Edge, **21**, No. 3, 268–314.

Silva, A., Neto, I. L., Carrasquilla, A., Misságia, R., Ceia, M., and Archilha, N., 2014, Neural network computing for lithology prediction of carbonate- siliciclastic rocks using elastic, mineralogical and petrographic properties, 1055–1058.

Smith, K., 2017, Machine learning assisted velocity autopicking, 5686–5690.

Wrona, T., Pan, I., Gawthorpe, R. L., and Fossen, H., 2018, Seismic facies analysis using machine-learning: GEOPHYSICS, **0**, No. ja, 1–34.

Zhang, L., and Zhan, C., 2017, Machine Learning in Rock Facies Classification: An Application of XGBoost, 1371–1374.