AN APPLIED APPROACH TO PREDICTING PETROPHYSICAL LOG DATA WITH MICROSOFT ML.NET REGRESSORS

Ryan Banas[1]

[1.] **PetroRes Consulting**

ABSTRACT

Machine learning has been used in the field of petrophysics for a long time and is well-understood by petrophysicists. Recently, however, more sophisticated regressors and models have been made available through interfaces such as Python and Microsoft's ML.Net machine learning software library. The purpose of this paper is to provide a guide to access current machine learning libraries to predict log data through software, such as Interactive Petrophysics (IP), utilize the latest logistic (and classification) regressors to predict various types of petrophysical data, provide a quantitative evaluation of the prediction accuracy and repeatability of new regressors against well-known standards, and finally provide a method to evaluate the best initialization and hyperparameters for each regressor used.

Quantitative comparisons of the regression results to existing and known methods such as multi-linear regression (MLR), Domain Transfer Analysis (DTA), and simple neural networks (NN) are provided as a benchmark. This paper includes a workflow to predict petrophysical data, including methods to group data and recommended pre-processing steps before training models.

This paper is on applied petrophysics, computer science, and quantitative evaluation of the methods and means to predict petrophysical data. It includes a concise analysis of what current technology is available, how the latest regressors function, strengths and weaknesses of these regressors, and the best parameters to use to predict different data types.

Although other machine learning petrophysics papers have been published, this one is specific to the decision-tree and gradient-based regressors available in the public technology space (e.g., Light GBM, fast forest, online gradient descent, etc.) and that have been implemented in the Microsoft ML.Net library.

INTRODUCTION

Most practicing petrophysicists probably use *machine learning* regularly and have been doing so for many decades before the artificial intelligence renaissance that has recently taken place. Methods such as logistic regression and classification are routinely used to predict petrophysics-related data. The domain of petrophysics is vast and encompasses many different data in the form of wellbore wireline measurements (logs), seismic data, rock facies, core measurements, etc. to which these algorithms may be applied.

There are curve prediction tools available in most commercial petrophysics software such as linear regression, non-linear curve fitting, MLR, DTA, NN, etc. The limitations and accuracy of these predictors are sufficiently understood from decades of use. This paper provides an overview of some newer technologies and quantification of decision tree and gradient-based regressors in comparison to these older and well-known methods of estimating data.

A basic workflow is introduced that outlines the necessary steps to use the newer technologies to predict data. Details are provided on a specific implementation of these new algorithms utilizing Microsoft's ML.Net library.

LOSS FUNCTIONS

A loss function quantifies the magnitude of error between predicted and actual values. Distinct types of loss functions can be used to help guide model fitting and assess the overall quality of model predictions.

Commonly used loss functions are the coefficient of determination (R^2), Mean Absolute Error (MAE), Mean

Square Error (MSE), and Root Mean Square Error (RMSE). These loss functions have different units, uses, and applications.

The coefficient of determination (R^2) is defined in Equations 1 to 5. It provides a measure of the strength and direction of association that exists between two variables (Doge, Y., 2008, pp. 88-91).

This loss function assumes the following: continuous variables, there is a linear or linearizable relationship, no significant outliers exist, and the variables are approximately normally distributed. It does not provide much information about overfitting issues or feature importance.

$$\bar{y} = \frac{1}{n}\sum_{i}^{n} y_i$$

Equation 1: mean value

$$e_i = (y_i - \hat{y}_i)$$

Equation 2: error or residual between actual value *y*, and predicted value $\hat{y}$

$$SS_{RES} = \sum_{i}^{n} {e_i}^2$$

Equation 3: sum of residual squares

$$SS_{TOT} = \sum_{i}^{n} (y_i - \bar{y})^2$$

Equation 4: sum of total deviation from mean

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}}$$

Equation 5: coefficient of determination

Mean Absolute Error (MAE) (Equation 6) treats all errors similarly, with no penalty applied to outliers. This may be a better metric for noisy data. The units are also easy to interpret and understand (Ciampiconi et al., 2023).

$$MAE = \frac{1}{n}\sum_{i}^{n} |e_i|$$

Equation 6: mean absolute error

Mean Square Error (MSE) (Equation 7) penalizes outliers with a square term. MSE is also scale-dependent and has different units than the original function, making it more difficult to directly compare (Dodge, Y., 2008, p. 138).

$$MSE = \frac{1}{n}\sum_{i}^{n} (e_i)^2$$

Equation 7: mean square error

Root Mean Square Error (RMSE) (equation 8) takes the root of MSE thereby making the units more interpretable while still penalizing outliers (Dodge, Y., p. 366).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i}^{n} (e_i)^2}$$

Equation 8: root mean square error

Loss functions can drive machine learning model training through iterative minimization. It is important to choose the correct loss function to achieve the desired results.

MACHINE LEARNING

There are many different types of machine learning algorithms available through public software libraries. Some are suited to solving specific problems and may or may not be tangible to a petrophysicist working with subsurface data.

Three main types of machine learning models are implemented in Microsoft's ML.Net software library: logistic regression, classification (binary and multi-class), and neural networks (e.g., deep neural networks, convolutional neural networks).

Logistic regression estimates continuous functions (labels) using inputs (features). Classification can be of both binary and multi-class types, with the latter more appropriate for problems such as lithologic facies prediction. Neural networks have been successfully used to build language learning models (i.e., Chat Generative Pre-Trained Transformer) and models for image recognition, pattern matching, and audio processing. These are generally regarded as *black boxes* that are very vast and deep neural networks trained on enormous amounts of data.

Each one of these learning models have different applications that depend on the problem that is presented and the type of data that is used. Tradeoffs occur between the ability to handle the type of data (in terms of missing values, outliers, irrelevant inputs, interpretability of the model) and predictive power (Hastie et al., 2017, pp. 351-352).

The newer decision tree-based algorithms examined in this paper can handle both logistic regression and classification. In ML.Net, these include Light Gradient Boosted Machine (GBM) (Guolin et al., 2017), Fast Tree, Fast Tree Tweedie, Fast Forest, and Generalized Additive Model (GAM) (Hastie et al., 2017, pp 295-297). There are additional Quasi-Newtonian or iterative mathematical optimization algorithms such as Stochastic Dual Coordinate Ascent (SDCA) (Shalev-Shwartz, S., 2013), and Online Gradient Decent (OGD). These are methods that are used to find zeroes or local maxima and minima of differentiable loss functions thereby optimizing weights in a model for the best fit.

Additional linear models such as Ordinary Least Squares (OLS) and Poisson (Limited-memory Broyden-Fletcher-Goldfarb-Shanno, L-BFGS) regression are included.

Details of these algorithms can be found in the Microsoft Machine Learning .NET API documentation (https://learn.microsoft.com/en-us/dotnet/machine-learning/).

BASIC DECISION TREES

Decision trees are simple data structures represented by nodes and branches. Nodes can either be a decision node (internal) or a leaf node (external). Branches between nodes represent decision rules. Decision trees work by asking binary (yes/no) questions to split nodes and are well suited to regression and classification problems (Hastie et al., 2017, pp. 307-310).

Decision trees operate based on two principles: entropy and information gain.

Entropy (E) is a mathematical quantification of *randomness* in a data set. It is defined in Equation 9 for N classes with a probability (p_i) of randomly picking an element of class i. P_i, represents the number of members in class i divided by the total number of members of all classes.

$$E = -\sum_{i}^{N} \mathrm{p}_i \log_2(\mathrm{p}_i)$$

Equation 9: entropy

Information Gain (IG) quantifies the quality of splitting a data set at a specific point by computing the amount of entropy removed. When building decision trees, it is advantageous to seek zero entropy in each branch to create more accurate predictions. This essentially quantifies the process of splitting the data into well-sorted sets.

Information gain in a decision tree is calculated by subtracting the weighted average of the entropy values computed for each child node. The weighting term, γ represents the number of elements in each child node divided by the total number of elements in all child nodes.

$$IG = E(parent) - \sum_{i}^{N} \gamma_i * E(child_i)$$

Equation 10: information gain

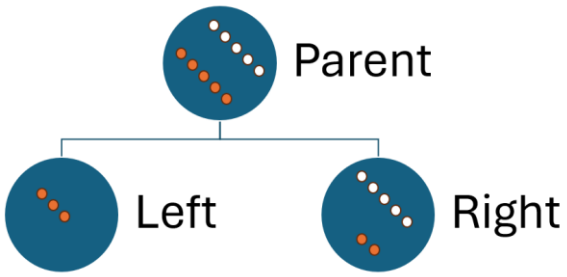


Figure 1: parent and child nodes in a tree

As an example, the entropy of the parent node of Figure 1 is equal to 1.0 (equivalent number of elements in each class, Equation 9), and a split is performed that results in two branches. The left branch has a single class (entropy = 0, Equation 9) with 3 elements, and the right branch results in 7 elements of two different classes.

Figure 2: entropy calculations

The entropy value of the right branch computed using Equation 9 (as in Figure 2) is 0.8631. In this case, the weight terms are 0.3 (3/10) for the left branch (γ_{left}) and 0.7 (7/10) for the right branch (γ_{right}). The resultant value of information gain (from Equation 10) is 0.3958 as shown in Equation 11. This value of information gain represents how much entropy is removed from the system due to the split.

$$\begin{aligned} IG &= 1 - \left[E_{left}\gamma_{left} + E_{right}\gamma_{right}\right] \\ &= 1 - [0 * 0.3 + 0.8631 * 0.7] \\ &= 1 - 0.6042 \\ &= 0.3958 \end{aligned}$$

Equation 11: information gain

Higher values of information gain imply more sorting has occurred. In decision trees, the amount of IG can also be used to determine how useful a particular feature is sorting classes. The best decision trees have attributes that return high information gain and low entropy.

Figure 3 shows an example of utilizing wireline log measurements as features in building a decision tree. Cutoff values can be used as decisions to sort the data as depicted in the branches and nodes of the tree.

Overfitting Problem

Due to the inherent nature of decision tree algorithms, there is a tendency to overfit data while training. This occurs when the tree recursively splits the feature space creating many branches and leaf nodes leading to extremely specific rules that only apply to the training data set.

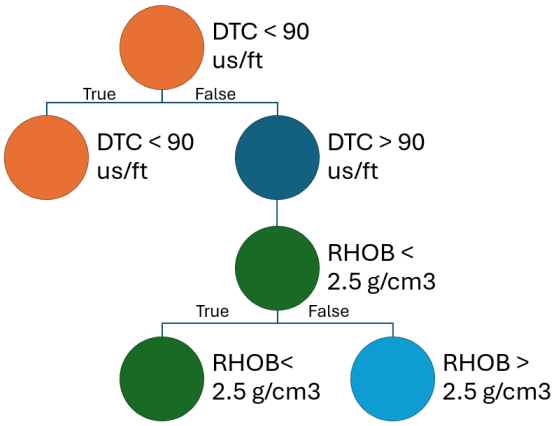


Figure 3: wireline log-based decision tree

An over-fit model tends to capture noise, is fit-for-purpose, overly complex, and sensitive to inputs (Hastie et al., 2017, pp.219-220).

There are strategies or techniques to counter this effect, including regularization, pruning, controlling tree depth, and ensemble learning.

Other Notable Problems with Decision Trees

Decision trees can also suffer from instability due to high variance. Since trees are hierarchical structures, errors made in the initial splits tend to propagate down the tree. They are also prone to producing data with a lack of smoothness (Hastie et al., 2017, pp. 312-313). Techniques discussed later in this paper, such as bagging, boosting, and controlling tree growth parameters help mitigate this problem.

Ensemble Learning

Ensemble learning is a technique that enhances prediction accuracy by utilizing multiple models in conjunction. This approach creates a sophisticated model by integrating a variety of simpler models (*weak learners*) (Hastie et al., 2017, pp. 605-606).

A random forest (Figure 4) is a type of ensemble learning algorithm where several decision trees are created in parallel with subsets of the training data. An average of the results of each model is taken as the final answer from the ensemble (Hastie et al., 2017, pp. 587-588).

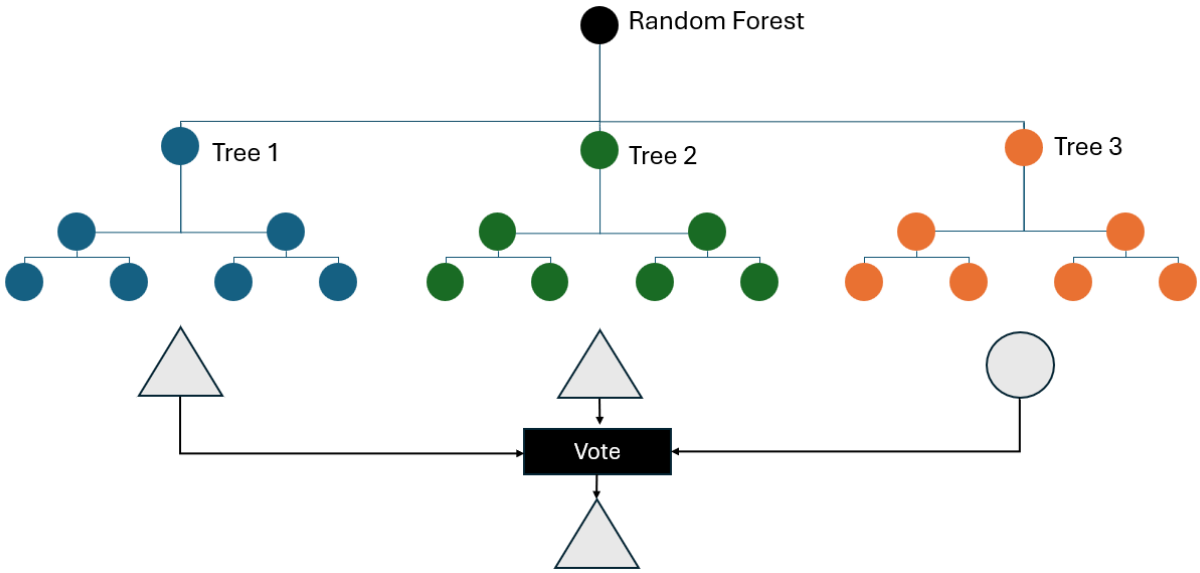


Figure 4: depiction of random forest

There exists a variety of ensemble learning models. The reader may be familiar with terms such as bagging and boosting. Bagging (or bootstrap aggregation) refers to the process of training multiple models in parallel, as implemented in a random forest (Hastie et al., 2017, pp. 249-252, 282-288). Bagging tends to be more flexible and less sensitive than boosting. Bagging utilizes random sampling with replacement, and works well with decision trees (Hastie et al., 2017, pp. 587) (Doge, Y., 2008, pp. 51-53).

As opposed to bagging, boosting is the sequential training of multiple models whereby each subsequent model learns from the mistakes of the previous model. Subsequent learners place more emphasis on undertrained or misclassified samples thereby improving the model fit, however, this can lead to over-fitting. Boosting utilizes random sampling with replacement over weighted data. It is important to point out that data is weighted in a boosted model, unlike bagging. Boosting is used in gradient-boosted trees (Hastie et al., 2017, pp. 86-88, 337-342, 353-358, 605-606).

The process of boosting produces incremental gains that can be added together to create a model that is very accurate at predicting a value. This is done by combining the many *weak learners* into a single strong learner.

As an example, the first model (*m*) in a boosted ensemble may predict a value by simply averaging all the values in a data set. To improve upon this, the subsequent model (*m+1*) will aim to minimize the residuals or differences between the predicted values and the actual values. Scaling the residuals with some factor helps to prevent over-fitting and to generalize the model. This scaling factor is defined as the *learning rate.*

By using a differentiable loss function, such as MSE, the process of gradient descent can then be used to help to minimize the residuals by numerical optimization. Hence, we have *gradient-boosted trees*. The gradient movement occurs in the direction of minimizing the loss function, i.e., a negative gradient. Predictions for each tree added to the ensemble model are multiplied by the learning rate. The learning rate thereby controls the step size the ensemble takes in converging. It is advisable to employ a small learning rate along with a substantial number of trees in boosting to facilitate easier convergence (Hastie et al., 2017, pp. 358-361).

As with other similar convergence algorithms, the number of iterations can be limited if the loss metric does not improve after many boosting rounds thereby stopping the algorithm and saving computational time.

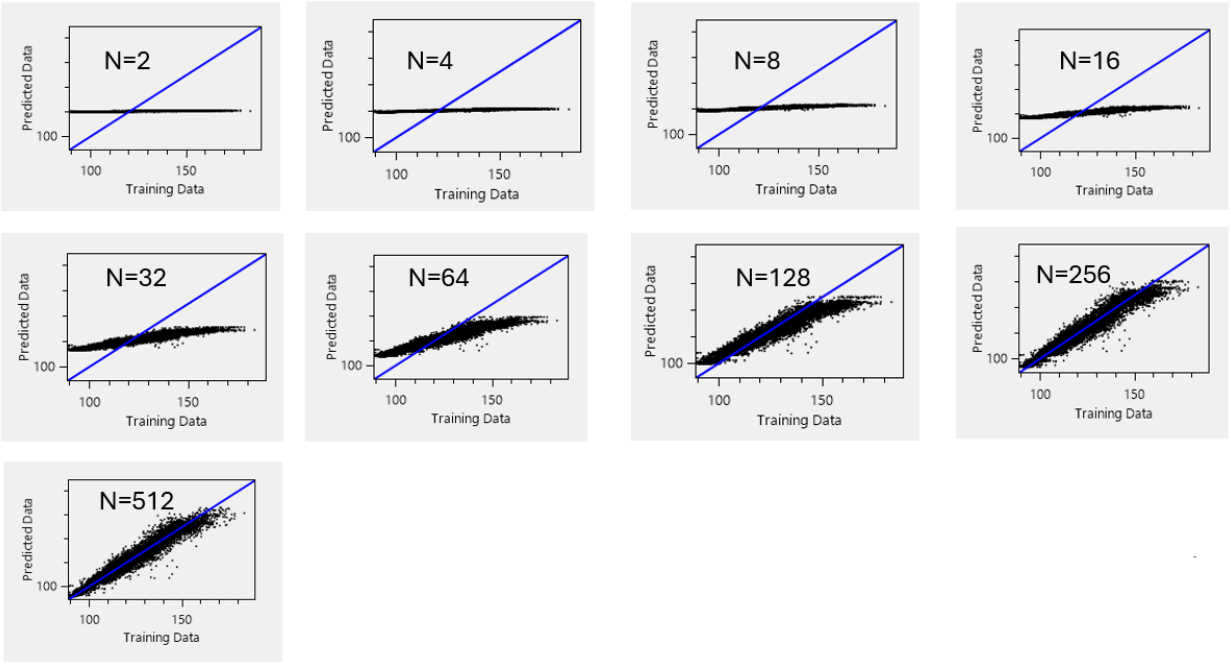


Figure 5: predicted data over *N* boosting rounds

Gradient boosting is well suited for tabular data (i.e., log or core data), may be used with both logistic regression and classification algorithms, and can be quantitatively interpreted unlike other machine learning black box models (i.e., DNN/CNN).

An example is shown in Figure 5 of training and predicted data with Light GBM over *N* boosting iterations. The average value of the training data is first predicted and as the number of boosting iterations (models added to the ensemble) increases the predictions become more accurate.

There are several boosting algorithms available within ML.Net: Gradient-based One-Side Sampling (GOSS), Gradient Boosting Decision Tree (GBDT), and Dropouts meet multiple Additive Regression Trees (DART). These boosting algorithms have different performance characteristics and applications for specific types of data. For example, GBDT may be a better choice for clean data sets whereas DART may be better suited to noisier data (Soyoung, L., 2024).

LIGHT GBM

Light GBM is one of the most accurate algorithms available from ML.Net when predicting petrophysical data from the author's experience. It is widely used and well-maintained.

Light GBM has been optimized to create decision trees that grow leaf-wise. It uses histogram-based algorithms to bucket data into bins that are used to calculate information gain and create splits within the data set as the tree grows. Light GBM uses GOSS to improve training efficiency. It also employs feature bundling to combine features and reduce dimensionality in the data set. This results in more efficient tree construction (Goulin et al., 2017)(Soyoung, L., 2024).

Like other decision-tree-based algorithms, Light GBM suffers from the problem of over-fitting training data. To help mitigate this, parameters can be constrained that control the growth of the tree. Specifically, weak predictors are purposefully created (shallow trees) and controlled by hyperparameters to prevent over-fitting. Regularization also plays a large role to help control over-fitting.

Light GBM can be used to solve both logistic regression and classification problems.

ML.NET ALGORITHMS

Table 1 summarizes each ML.Net regressor, data structure, and algorithm used.

Regressor	Data Structure	Algorithm Details
Light GBM	Decision tree	Gradient Boosted Decision Tree (Goulin, K. et al., 2017)
Fast Forest	Decision tree	Random forest (Hastie et al., 2017, pp 587-588)
Fast Tree	Decision tree	Gradient Boosted Decision Tree (using the Multiple Additive Regression Tree algorithm) (Rashmi, K.V., 2015)
Fast Tree Tweedie	Decision tree	Gradient Boosted Decision Tree (Yang et al., 2016)
OLS	Linear least squares	Ordinary least squares linear regression
SDCA	Convergence algorithm	Stochastic Dual Coordinate Ascent optimization technique for convex objective functions (Shalev-Shwartz, S., 2013) (Hsiang-Fu, Y. et al., 2010)
OGD	Convergence algorithm	Stochastic gradient descent optimization technique for convex loss functions (non-batch)
LBFGS Poisson	Linear regression	Poisson regression implementation with LBFGS optimization technique (generalized linear model)
GAM	Decision tree	Shallow Gradient Boosted Trees (Lou, Y., et al., 2012) Non-linear functions

Table 1: summary of ML.net algorithms

EVALUATING ALGORITHMS

In this paper, the predictions from training and test data made by each algorithm are evaluated based on calculated loss functions. In addition, feature information is provided for the algorithms by permutation feature importance calculations. Five-fold cross-validation is performed on test data, and algorithm prediction repeatability is analyzed by varying hyperparameters during training.

Predicted data is also visualized graphically in cross plots to evaluate fit and overall shape. This is an important step as some algorithms can be prone to creating oddly shaped predicted data sets in these 2D spaces (e.g., flatline cutoffs at specific values, etc.) that may not be discernable in histograms or statistical plots.

Input data is split into a test and train fraction for evaluating each algorithm. For training, 80% of the data is used with the remaining 20% for testing. This helps the interpreter assess whether the model is overfitted to the training data. By testing the predictions with the test data set using k-fold cross-validation, a quantitative assessment of the generalization of the model is possible.

SOFTWARE IMPLEMENTATION

In this paper, Interactive Petrophysics and Microsoft's ML.Net library are used together to perform predictions of log data by a computer program written by the author in C#.

An Application Programming Interface (API) is made

available through IP to access objects within the memory space of a running instance of the application. Specifically, access is provided to array-based log data represented in single-precision floating-point 32-bit format. These data are imported into the ML.Net data structures to build regression models and to predict data. The predictions are then written back into the IP memory space via the API resulting in a new curve. This new estimated data can then be visualized and used for subsequent calculations.

The basic workflow of ML.Net consists of loading training data into an implemented data frame interface (*IDataView object*), providing options for an algorithm, creating a model, training it, making predictions, and evaluating the predictions. This workflow is performed iteratively until the user is content with the results.

The terminology used by Microsoft to describe the process of mapping the raw data to the resultant predictions is a *pipeline*. Pipelines are created for each algorithm and can be static or *sweepable.* A sweepable pipeline is a collection of pipelines that have been configured to vary options or parameters to provide multiple estimators.

In source code, the process consists of extracting the data from the data source (e.g., petrophysics software API, LAS file, database, etc.) and populating a data frame. Once this step is complete, the data is split into training and test sets based on a fraction specified by the user. A subroutine is provided via ML.Net to perform splits. The method in which this subroutine performs a split is repeatable such that each time it does so, the same resultant data sets are created from the original input data. Thus, no concern is needed for unrepeatable or random results in training over multiple passes.

The next step in the process is to create pipelines for specific algorithms with user-specified options and additional transforms (such as normalizers). Once a pipeline is configured, it can then be provided the training data and fitted resulting in a trained model that can make predictions. Test data is used to evaluate the fit and overall generalization of the model.

Permutation Feature Importance (PFI) is calculated with a subroutine implemented in the ML.Net library. The subroutine iteratively investigates each feature's contribution to the model (i.e., weight) by selecting a single feature and permuting its values a specified number of times. This is achieved by randomly shuffling the position of the values in the array. The newly permuted feature curve is then run through the model and the loss function is computed. This is done as many times as the user specifies, finally culminating in a quantitative assessment (i.e., by loss function) of how much the changes in a specific feature impacted the accuracy of the predictions. This process is then performed for each feature and the magnitude of loss function changes can be tabulated by feature. This methodology is algorithm-agnostic and is widely applied.

Unfortunately, the MLR, DTA, and NN modules of IP are not made available through the provided API and are therefore not accessible from the software written by the author. Due to a lack of time and resources, PFI is not performed on these algorithms as it would need to be done manually. In place of this, a similar feature selection module (experienced eye) is available within the IP software to assess the feature importance for these algorithms. Results are calculated for these three estimators as an indirect but meaningful comparison.

ML.Net provides a subroutine to the user (AutoML) that alters hyperparameters and configuration options for algorithms to optimize model training and fitting. AutoML can create numerous pipelines for different algorithms dynamically and execute them in a single run providing flexibility. This enables the user to try different algorithms on their data and determine which ones are the best suited. The AutoML subroutine aims to optimize a loss function specified by the user (e.g., R^2, MAE, etc.).

AutoML utilizes different searching algorithms such as grid search, random search, and Bayesian optimization to iteratively select hyperparameters and options as it builds pipelines.

Once the user is content with a trained model, the binary object representing the model in computer memory may be saved to disk as a zip file. It can be retrieved at a later point and used again to estimate data. It is noteworthy that not all information from the model training process is archived in the saved file. In fact, very little information is preserved about the training data, hyperparameters used, model options, etc. This issue arises from the design of the objects and interfaces within ML.Net.

A solution proposed by the author is to create a serializable wrapper class around the byte array data representing the model. Additional important information can then be stored in this class and saved and retrieved as a serializable object. Unfortunately, it is not

possible to retrieve some information from the model runs generated by AutoML pipelines such as hyperparameters used. L1 and L2 regularization parameters may be extracted from logs but it is a tedious and difficult process.

The software created by the author implements a queueing system where parameters for specific runs can be set up sequentially, including the selection of training data to aggregate by formation top or by facies curves and then run together at the same time. This allows the user the ability to automate predictions by configuring labels, features, training data, test data, algorithms, and hyperparameters at the same time.

It is beyond the scope of this paper to provide comprehensive source code detailing an implementation.

HYPERPARAMETERS

Hyperparameters generally refer to a set of specific attributes that are related to the building of machine learning models. These attributes commonly include parameters that control things like regularization. In the case of tree-based algorithms: tree size, criteria for splitting nodes, etc.

Hyperparameters play a large role in the outcome of a model being trained and how it will predict data. Below are some key hyperparameters related to decision trees, how they are used, and how they affect fitting.

Regularization Parameters, L1 and L2

Decision trees are designed to over-fit data when training. One method of countering this effect is to implement regularization. There are three types of regularization: Least Absolute Shrinkage and Selection Operator (LASSO, L1), Ridge (L2), and by utilizing both L1 and L2 (ElasticNet) (Hastie et al., 2017, pp. 61-73, 82, 364-367, 607-611).

Regularization works by adding a penalty term to the loss function (i.e., MAE) per Equation 12 (in terms of L1) (Ciampiconi et al., 2023).

$$Loss_{L1} = Loss_{ori} + \lambda \sum |w_i|$$

Equation 12: L1 regularization

In this case, the parameter λ determines the strength of the regularization against the coefficients (weights) of the model (w_i). This encourages the model to keep those coefficients as small as possible.

The L1 or LASSO method of regularization tends to drive some of the weights to zero, which can help remove non-contributing features in high-dimension feature sets, thereby performing feature selection for you.

L2 or ridge regularization works the same as L1, however, the penalty applied uses squared weights which helps balance out features instead of driving them to zero (Equation 13). This also helps to enhance the stability and performance of a model.

$$Loss_{L2} = Loss_{ori} + \lambda \sum |{w_i}^2|$$

Equation 13: L2 regularization

There is a bias-variance tradeoff between L1 and L2 regularizations regarding prediction error. L1 tends to have a higher bias and lower variance (less complex model), whereas L2 has a lower bias and higher variance (more complex model) (Hastie et al, 2017, p. 38, pp. 219-227).

In summary, L1 regularization helps remove useless features and promotes sparsity, L2 will retain features but try to balance them.

ElasticNet regularization linearly combines both L1 and L2 penalties.

If regularization results in very large λ values it will produce an under-fit model, whereas if the λ values are small it will produce an over-fit model.

Learning Rate

Another important hyperparameter for decision trees is the learning rate. This controls the contribution of each weak learner in an ensemble-boosted model. Slower learning creates a more robust model less prone to over-fitting and is probably one of the most important hyperparameters for the user to consider with boosted ensemble models.

Number of Iterations

The number of iterations parameter controls how many predictors are made in the ensemble model (number of subsequent trees in the case of boosting). There is a tradeoff between the number of iterations and learning rate. If the learning rate is low, the number of iterations needs to be high. It is possible that too many iterations can lead to overfitting a model.

Maximum Depth, Maximum Leaf Nodes

These parameters control the size of the tree, complexity, and help to control over-fitting.

Minimum Samples Per Leaf

This parameter controls the minimum number of samples used to create a split to compute information gain. Increasing the value of this parameter may help to prevent over-fitting. This parameter guarantees a minimum number of samples in a leaf node.

Minimum Samples Per Split

This parameter controls the minimum number of samples required to create a split. This helps prevent splits that may result in very small datasets. Larger values may help to prevent over-fitting, like minimum samples per leaf. However, this parameter differs as it does not guarantee a minimum number of samples in a leaf node.

Feature Fraction

This parameter controls the percentage of features selected randomly at each iteration when building trees. Higher values may help to prevent over-fitting.

Bagging Fraction

When bagging, this controls the fraction of rows selected randomly.

Minimum Gain to Split

This parameter specifies the minimum information gain required to split a leaf.

Early Stopping Round

This parameter controls the number of training rounds that occur. If a loss metric does not improve in the last *x* rounds, then model training is stopped.

Subsampling Rate

This parameter controls the portion of data used to train each tree (i.e., training rate).

DATASET

A Permian Basin well (University Lands 7-16 #10) is chosen from the public domain to use in this paper. This well has wireline log data available with total gamma ray (GR), bulk density (RHOB), photoelectric factor (PEF), thermal neutron porosity (NPHI), deep laterolog resistivity (RDEEP), and compressional sonic slowness (DTC). This well is selected as these are commonly available wireline measurements and provide a good application of logistic regression.

The curve chosen to predict is PEF, known to be a difficult log measurement to estimate as it varies non-linearly with other log measurements. In addition, numerous historical logging suites without PEF measurements are widely available. This commonly requires synthesis of PEF data to perform interpretations. All other wireline curves listed above are used as features in predicting PEF as summarized in Table 2.

Features	Label
GR (GAPI)	PEF (b/e)
RHOB (g/cm^3)	
NPHI (dec.)	
DTC (µs/ft)	
RDEEP (ohm-m)	

Table 2: data used for logistic regression

The University Lands 7-16 #10 well penetrates Permian stratigraphy from the Glorieta formation through to the Strawn. Various lithofacies are present including limestones, dolostones, sandstones, silts, organic-rich shales, carbonaceous shales, etc. This provides a robust test of the prediction algorithms over a wide range of log responses in each distinct lithology (Figure 6).

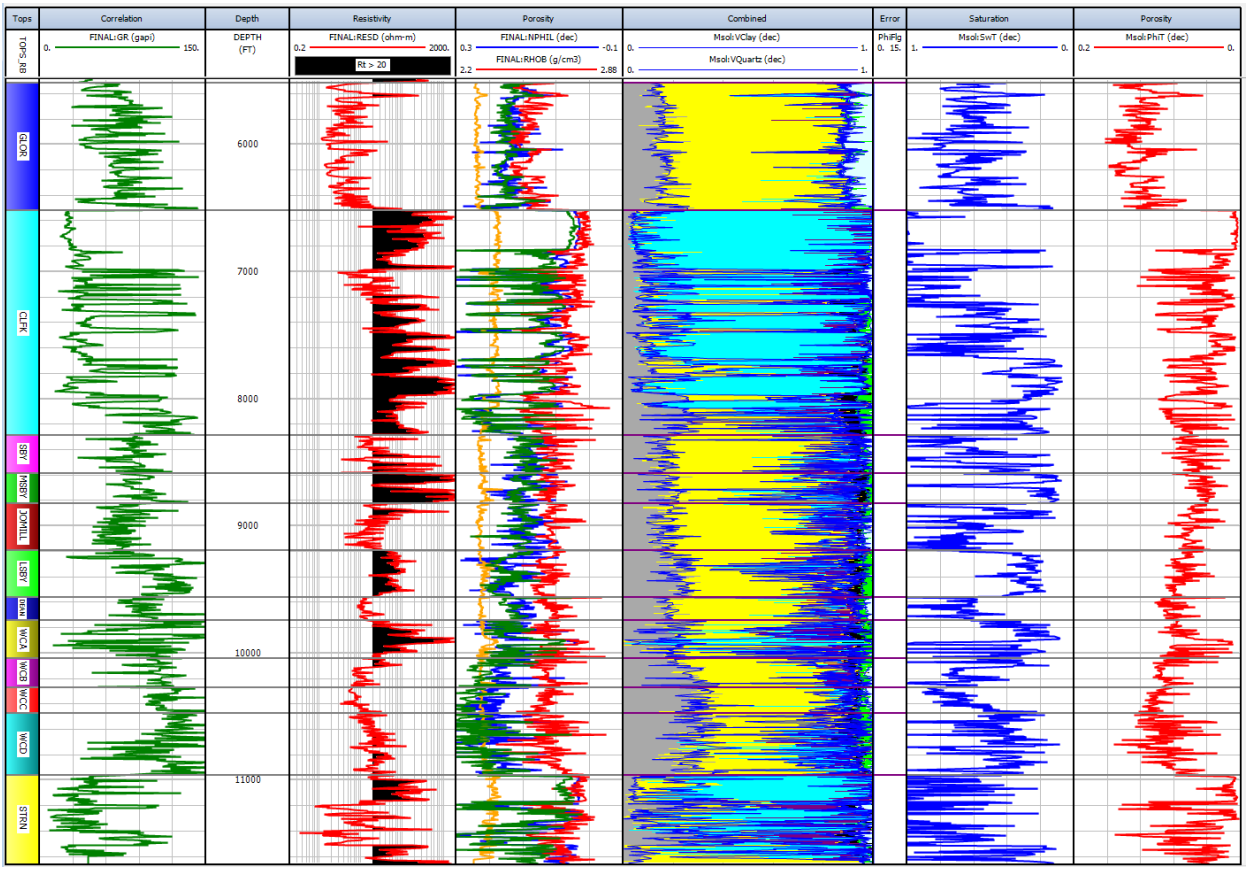


Figure 6: University Lands 7-16 #10 wireline data

GROUPING DATA

As a standard in this paper, data has been grouped for regression over the entire well interval from top to bottom. It has, however, been observed by the author that if rock types are broken out and similar lithologies grouped together by facies and/or formations it improves the fit of the regressions. Training logistic regressions on formation or facies-grouped data is therefore recommended to get the most accurate results possible.

DATA PREPARATION

Prior to building machine learning models, data is prepared from the well by depth shifting all curves to the total gamma-ray as a reference and editing the log data using an automated MLR-based editing algorithm (Banas et. al., 2021).

The term *normalization* may have multiple meanings to a petrophysicist. One definition is to transform data on a well-to-well basis. This type of normalization has not been performed on this data set as the study consists of a single well and it is not necessary to do so.

Normalization in the domain of machine learning may refer to pre-processing data in the form of mapping values to a specified range suitable for ML models (i.e., 0-1). The author has performed this step, tested it against ML.Net's algorithms, and found that it is not necessary. No improvement or change in fits or loss functions has been noted by normalizing data to a 0-1 range. When using AutoML, however, some normalizers are automatically added to appropriate pipelines, making this step unnecessary for the user.

Another form of data pre-processing is linearizing specific features for use as training data. As an example, deep resistivity can be linearized by taking a logarithmic base 10 transform of the data. The author has tested the impact of linearizing resistivity in various algorithms when it is being used as a feature and found that it is only

impactful when running linear algorithms (i.e., MLR). Decision-tree and convergence algorithms do not seem to benefit from linearizing resistivity as a feature.

When resistivity is being used as a label (or curve to predict), it is beneficial to linearize resistivity for all algorithms. This has been tested by the author and thought to be to be very impactful in prediction accuracy. Since the base 10 logarithm of resistivity is being predicted, it is then necessary for the user to convert the predicted values back into the correct base (i.e., 10^x).

Environmental corrections have been found to be useful. This is especially the case in heavily washed-out intervals where highly weighted features can be environmentally corrected thereby improving the model. Environmental correction is also important in multi-well modeling as it assists in normalizing data.

METHODOLOGY

PEF is selected as a label to be predicted from the features GR, RHOB, DTC, NPHI, RDEEP (Table 2).

AutoML is used for all algorithms (except for GAM and any internal IP modules). A 10-second training time is used in the AutoML subroutine to optimize algorithm hyperparameters. The same test/train split fraction and resultant data sets are used throughout the process for all algorithms.

As the IP modules (i.e., DTA, NN, MLR) are not able to use data directly from the ML.Net application, the test and train data sets are written out as array curves into the IP software and then used as inputs for training.

Loss functions are computed for each model and tabulated to assess training and generalization (test) error over the data sets. In addition, for ML.Net models Permutation Feature Importance is computed and 5-fold cross-validation is performed on the test data set (Hastie et al., 2017, pp. 241-249).

MULTI-WELL WORKFLOW

When building machine learning models with data from multiple wells additional care must be taken to prepare the data properly for training.

The same workflow as previously described can be used consisting of depth shifting, editing, and environmentally correcting the data.

When working with multiwell datasets, however, the interpreter should evaluate the statistical relationships between well data when performing these steps to identify and correct any biases, shifts, potential outlying data, or noise.

Due to the ability of decision-tree-based regressors to fit data well, if noise or outliers are present in the training data set, the model will train to it and predict these values.

Well-to-well normalization should take place on feature and label data where appropriate. Widely applied normalization on unedited and un-environmentally corrected data can be detrimental in removing reservoir response and skewing data. This step needs to be performed with great care by the interpreter on a lithology or formation-based level with a concise understanding of the reservoir response.

Finally, once all these steps have been taken the data can be aggregated from many wells to train a model and predictions made in many wells with selected feature curves.

Multiple models can be built and deployed as needed to constrain correlations to specific lithologies or reservoir types. Furthermore, specific wells can be used for training purposes in a hub-and-spoke fashion to propagate predictive models to wells located close to the training well(s) with assumed similar properties.

This workflow is useful when generating synthetic data (i.e., PEF) by leveraging relationships in offset wells where data is available from specific formations or lithology types to create robust correlations. This is a better option than using data from lithologically dissimilar intervals within the same well to train a model.

Predicted data can be adjusted through normalization to reference wells or additional editing to produce the appropriate results as expected by the interpreter.

RESULTS

The tabulated results of estimating PEF with the different regression algorithms are shown in Table 3. The algorithm with the best fit, the highest R^2, and the lowest MAE is Light GBM. This algorithm has also been used in winning solutions of many machine learning competitions that have taken place (https://github.com/Microsoft/LightGBM/tree/master/examples).

Algorithm	Training Set R^2	Test Set R^2	Training Set MAE (b/e)	Test Set MAE (b/e)	Training Set MSE	Train R^2 % < Light GBM	Train MAE % > Light GBM
Light GBM	0.82520	0.84721	0.26090	0.24785	0.13829	0.00000	0.00000
Fast Tree	0.82252	0.84642	0.26295	0.24850	0.14047	0.32501	0.78533
Fast Tree Tweedie	0.82099	0.84101	0.26494	0.25266	0.14168	0.51042	1.54805
Fast Forest	0.79171	0.80245	0.29628	0.28954	0.16486	4.05852	13.55990
GAM	0.77474	0.77578	0.30778	0.30663	0.17854	6.11462	17.96800
DTA	0.74159	0.76590	0.32890	0.31500	0.20760	10.13210	26.05970
MLR	0.66506	0.66911	0.40630	0.40720	0.26550	19.40540	55.72539
OLS	0.66411	0.66912	0.40654	0.40751	0.26590	19.52110	55.82006
LBFGS	0.62690	0.62525	0.44075	0.44640	0.29541	24.03000	68.93043
SDCA	0.60657	0.61065	0.45290	0.45951	0.31148	26.49400	73.58609
NN	0.57610	0.58165	0.49930	0.50000	0.38230	30.18590	91.37014
OGD	0.43324	0.44593	0.56109	0.56177	0.44904	47.49840	115.0563

Table 3: PEF estimation results

The next best regressors are other ensemble decision-tree-based models such as FT, FTT, FF, and GAM. DTA then follows with MLR/OLS. Finally, NN and gradient-based optimization algorithms perform the poorest.

As Light GBM is the most accurate predictor, two calculations are made evaluating the training data MAE and R^2 values from other algorithms as compared to Light GBM. These are shown in the last two columns of Table 3. These calculations show that the other boosted decision tree-based algorithms are very similar in prediction accuracy, within 0.5% R^2 and 1.5% MAE. The full range of MAE for training, test data, and cross-validation is shown in Figure 7. As cross-validation is not performed with DTA, MLR, or NN, these results are omitted.

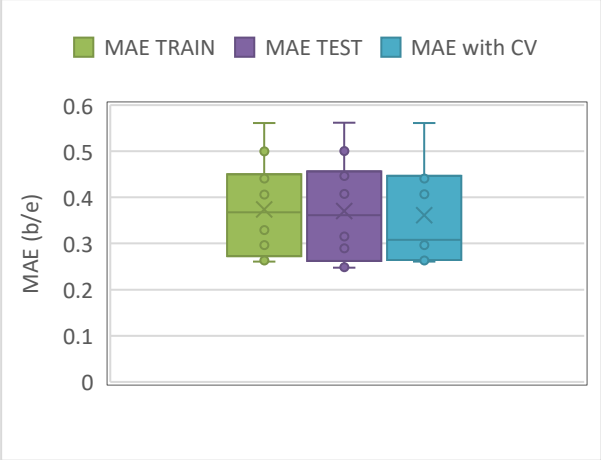


Figure 7: MAE for training, test and cross validation data

The MAE between test and train data is remarkably similar, shown in Figure 8.

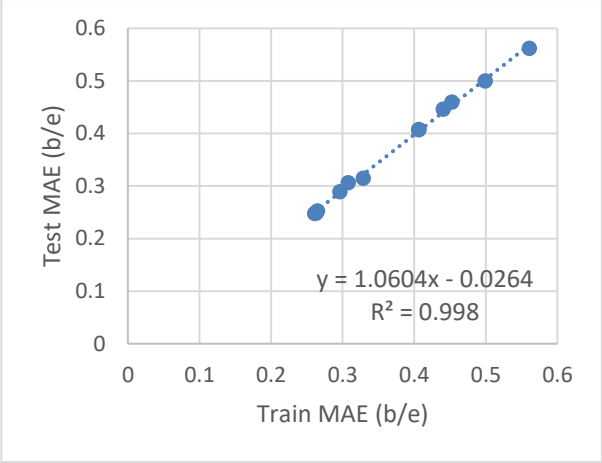
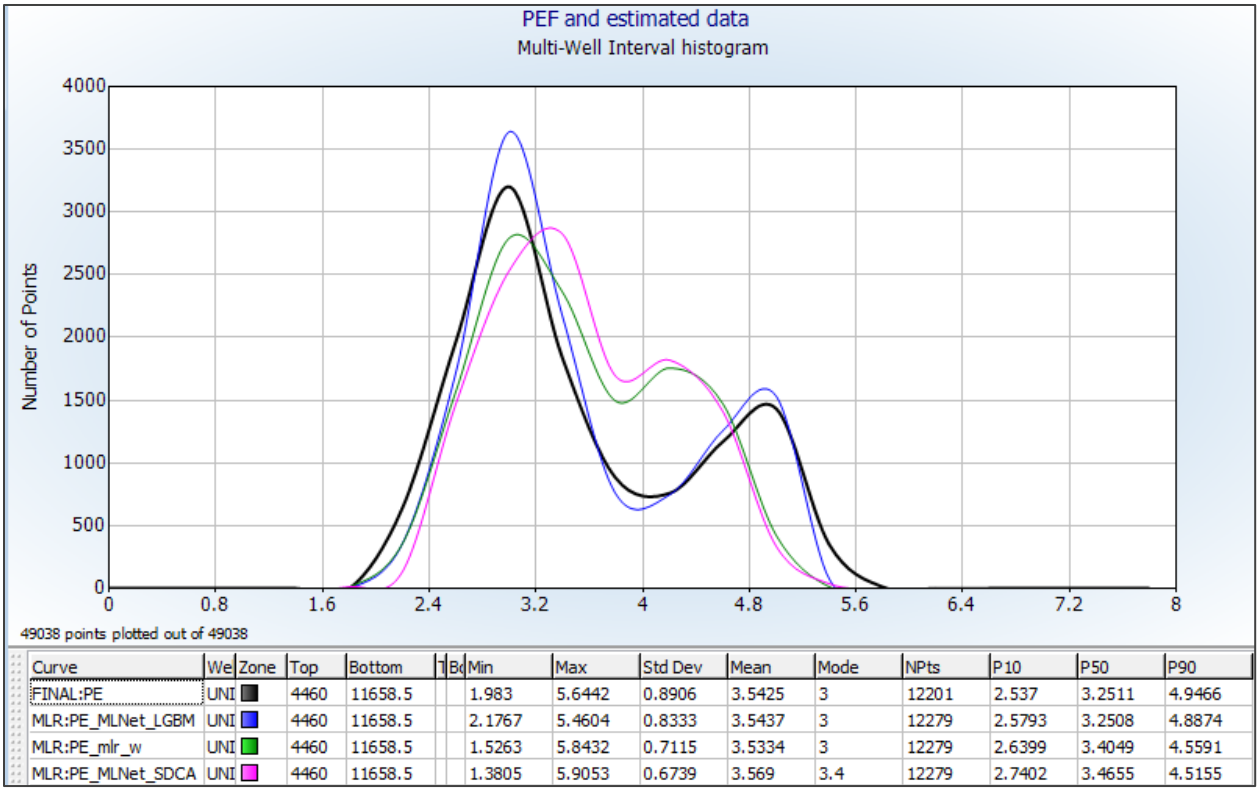


Figure 8: test and train MAE for all algorithms

Plotting the original recorded PEF data along with several estimated curves in different formats provides more insight about the overall accuracy of the predicted data.

As an example, the recorded PEF data over the entire well interval is displayed along with the estimated values from Light GBM, MLR, and SDCA in a histogram (Figure 10). These three algorithms are compared as representative estimators of a decision-tree algorithm, optimization algorithm, and multiple linear regression.

It is apparent that the MLR and SDCA predicted values are less capable of recreating the two modes in the



Curve	We	Zone	Top	Bottom	T	Bc	Min	Max	Std Dev	Mean	Mode	NPts	P10	P50	P90
FINAL:PE	UNI		4460	11658.5			1.983	5.6442	0.8906	3.5425	3	12201	2.537	3.2511	4.9466
MLR:PE_MLNet_LGBM	UNI		4460	11658.5			2.1767	5.4604	0.8333	3.5437	3	12279	2.5793	3.2508	4.8874
MLR:PE_mlr_w	UNI		4460	11658.5			1.5263	5.8432	0.7115	3.5334	3	12279	2.6399	3.4049	4.5591
MLR:PE_MLNet_SDCA	UNI		4460	11658.5			1.3805	5.9053	0.6739	3.569	3.4	12279	2.7402	3.4655	4.5155

Figure 9: PEF histogram

data set centered around 3 and 5 barns/electron, corresponding to shale and carbonate facies. Light GBM very accurately predicts the same modal behavior as the original data.

The data are displayed in a cross-plot against another variable (GR) in Figure 10. The same data and coloring scheme from Figure 9 are used to show the recorded and estimated PEF data. An outline of the recorded data is made in black. It is visible that Light GBM (blue) does an excellent job of matching the overall shape of the data trend in this 2D space, whereas the MLR (green) and SDCA (pink) predictions are skewed.

PFI results are computed in terms of R^2 change to the resultant regression for the ML.Net algorithms utilizing the test data set. An example of the PFI results and cross-validation for Light GBM is shown in Figure 11. This is performed for all ML.Net algorithms and the resultant PFI values per feature are aggregated into a box plot (Figure 12) to summarize the variability of each feature per algorithm.

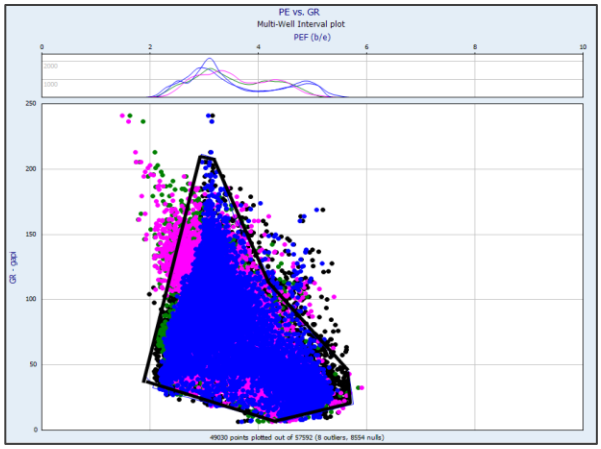
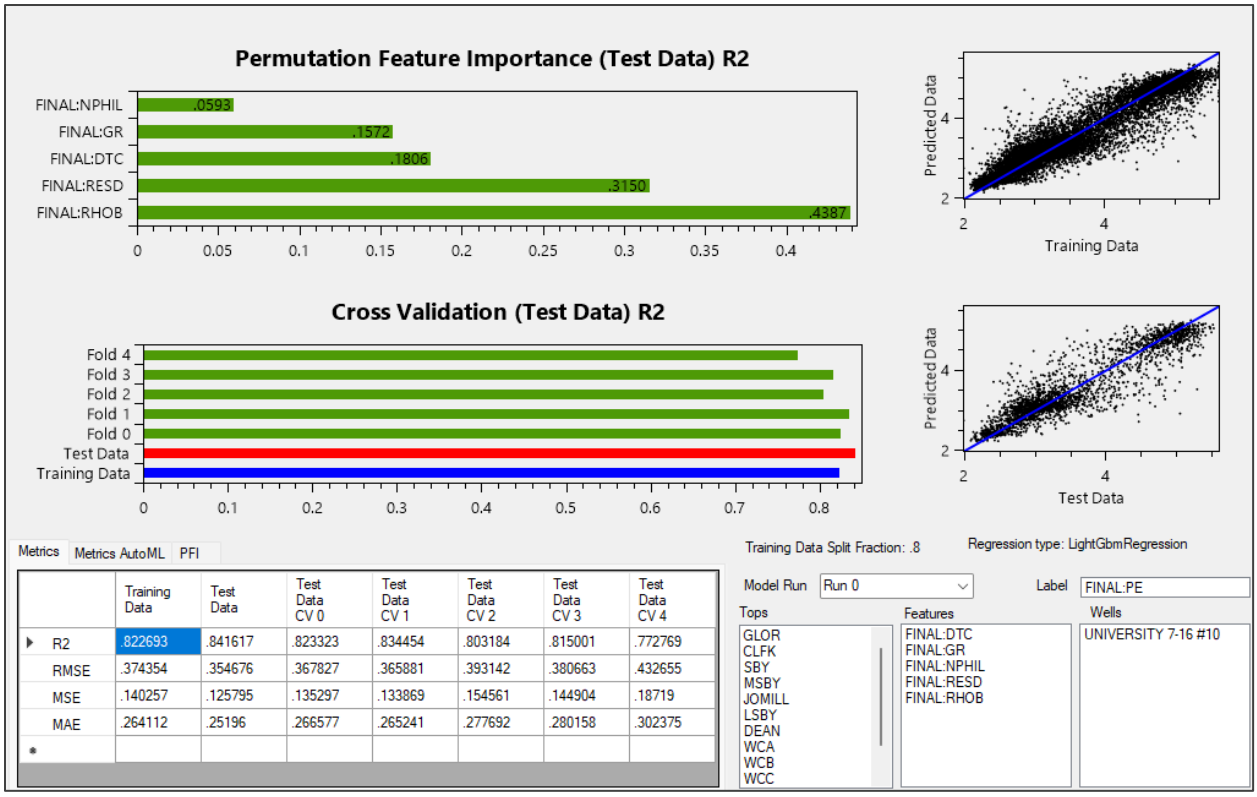


Figure 10: cross-plot of PEF data vs. GR

The results demonstrate that the largest variance is in RHOB, which happens to be the most important feature in most algorithms to predict PEF. Across all algorithms, generally, it follows that the order of importance of features in predicting PEF are (from most to least important): RHOB, RDEEP, GR, DTC, and NPHIL.



	Training Data	Test Data	Test Data CV 0	Test Data CV 1	Test Data CV 2	Test Data CV 3	Test Data CV 4
R2	.822693	.841617	.823323	.834454	.803184	.815001	.772769
RMSE	.374354	.354676	.367827	.365881	.393142	.380663	.432655
MSE	.140257	.125795	.135297	.133869	.154561	.144904	.18719
MAE	.264112	.25196	.266577	.265241	.277692	.280158	.302375

Figure 11: Example PFI and CV calculations for LGBM

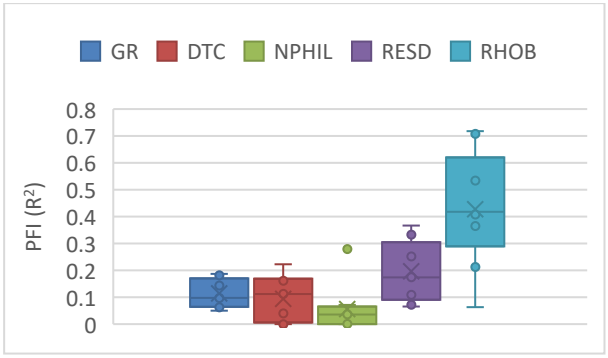


Figure 12: PFI results

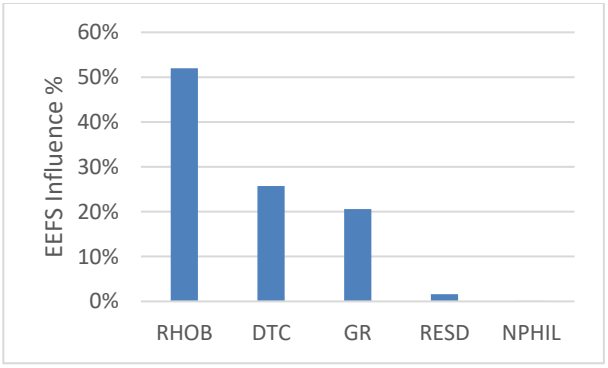


Figure 13: EEFS influence per feature

As a meaningful but indirect comparison of PFI for the DTA, MLR, and NN algorithms, the experienced eye (EE) module from IP is run. An 80/20 test/train data split is specified in the module. This is not necessarily the same split as created by ML.Net, as it is generated by the experienced eye module. The results are in terms of overall feature influence in a percentage value and shown in Figure 13.

It is worth noting that NPHI is identified through the EE feature selection module and the ML.Net PFI subroutine as a non-contributing feature. Based on these results the interpreter may choose to remove NPHI as a feature and not use it to predict PEF.

To understand the variance in outcomes from changing hyperparameters during the AutoML runs, MAE is tabulated for every pipeline created by AutoML for each

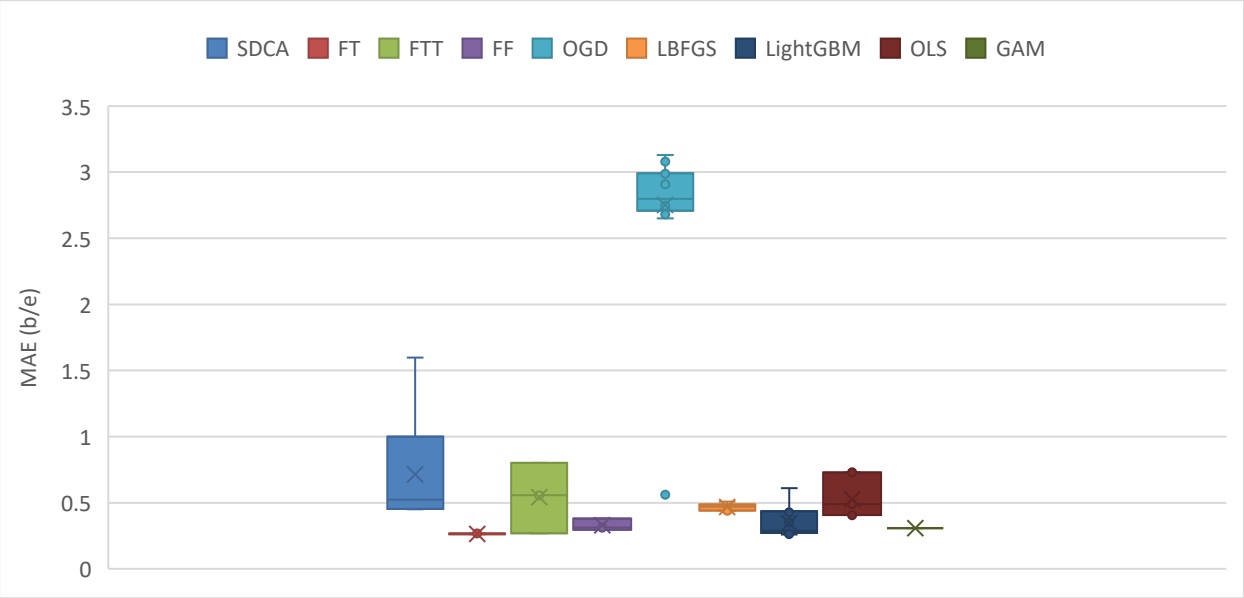


Figure 14: AutoML MAE variation

algorithm during model training. A training time of 10 seconds is used; however, this does not guarantee that each algorithm has an equal number of pipelines generated by AutoML, due to each one having different training speeds. The results are displayed in Figure 14.

It is observed the largest variance occurs in OGD followed by SDCA, FTT, OLS, Light GBM, FF, LBFGS, and FT. A GAM value is displayed, however, only a single pipeline is created as this algorithm is not useable with AutoML so there is no variation. It does appear that the decision tree-based algorithms have a reasonable range of MAE from this process of searching for hyperparameters by AutoML.

AUTOML REPEATABILITY (LIGHT GBM)

Several AutoML experiments and pipelines are created with different training times and hyperparameters to investigate the repeatability of the Light GBM algorithm. These data are examined to see how these variables impact the variance in the model fitting and estimations.

Ten runs are made in total with training times varying from 1-20 seconds. The results are displayed in Table 4 with R^2, MAE loss functions, and PFI values for features (in terms of R^2 change). Variability in R^2 and MAE of the training data set are summarized in Figures 15 and 16 respectively.

Figure 15: R^2 repeatability (training data)

Figure 16: MAE repeatability (training data)

Run #	Training Time (s)	R^2 Training Data	MAE (b/e) Training Data	PFI NPHI (ΔR^2)	PFI GR (ΔR^2)	PFI DTC (ΔR^2)	PFI RDEEP (ΔR^2)	PFI RHOB (ΔR^2)
1	5	0.8244	0.2614	0.0629	0.1628	0.1677	0.3397	0.4334
2	5	0.8232	0.2619	0.0922	0.2088	0.2613	0.3473	0.4460
3	5	0.8195	0.2666	0.0469	0.1452	0.1411	0.2835	0.4408
4	20	0.8236	0.2628	0.0598	0.1680	0.1754	0.3277	0.4745
5	5	0.8225	0.2645	0.0782	0.1502	0.2030	0.3121	0.3899
6	5	0.8232	0.2638	0.0392	0.1494	0.1641	0.2915	0.4174
7	10	0.8218	0.2644	0.0516	0.1648	0.1609	0.2934	0.4269
8	1	0.8185	0.2679	0.0606	0.1555	0.2086	0.2818	0.4533
9	3	0.8204	0.2660	0.0343	0.1424	0.1727	0.2534	0.3842
10	3	0.8185	0.2679	0.0620	0.1538	0.2102	0.2735	0.4454

Table 4: Light GBM repeatability

PFI repeatability is summarized per feature in Figure 17.

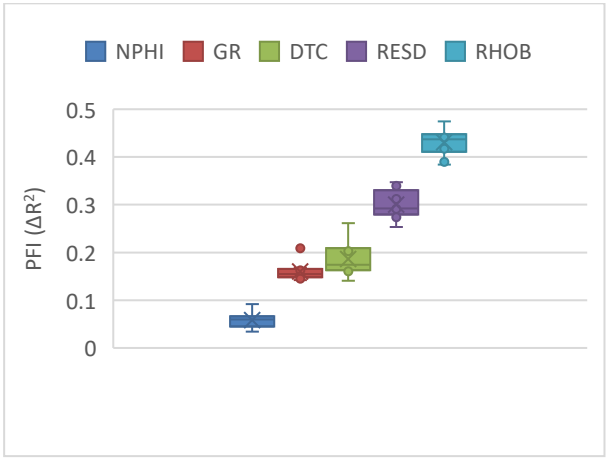


Figure 17: AutoML PFI repeatability Light GBM

To check Light GBM's sensitivity to noise, a random error is added to feature and label data. This is achieved by using a random function to select some rows as a percentage of the total and replace the values in these rows. In this case, 20% of the rows are replaced with random values. A random double number is then generated for each row between 0 and 1 and multiplied by the range of the PEF data and this value is then added to the minimum value observed as in Equation 14.

$$value = rand(x) * range + min$$

Equation 14: generating random data

The results of training using randomized data are shown in Figure 18. There is an increase in MAE with error as expected but variance does not change significantly.

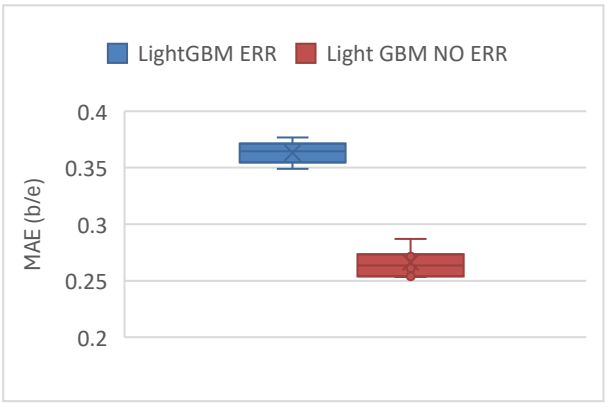


Figure 18: MAE with and without random error

Changes in PFI are also tabulated between the training data set with random error and the original. There is some change but not significant as seen in Figure 19.

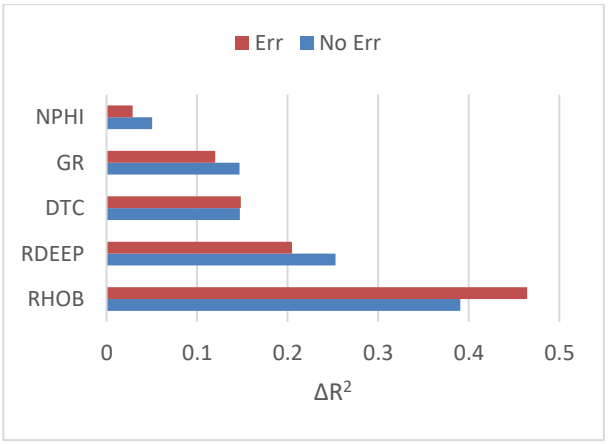


Figure 19: PFI changes with random error

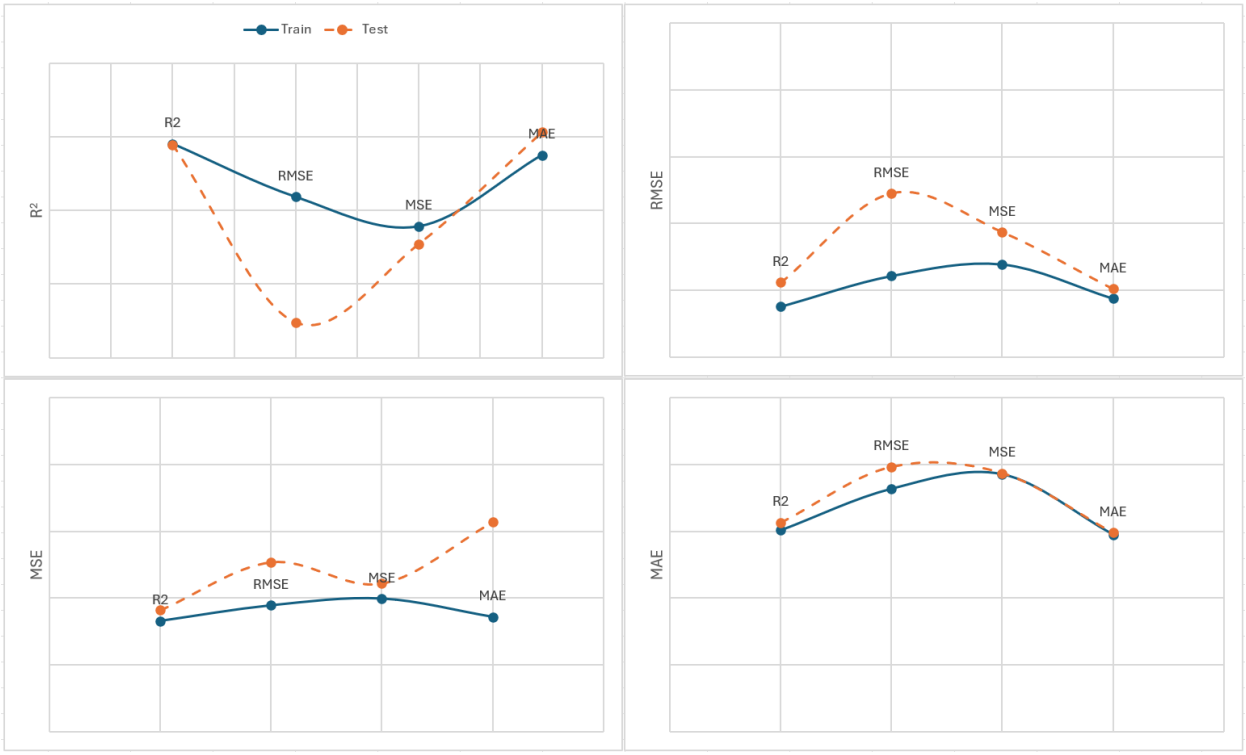


Figure 20: AutoML sensitivity to loss functions

AUTOML SENSITIVITY TO LOSS FUNCTIONS

Multiple runs are made with AutoML optimizing different loss functions to evaluate the sensitivity of AutoML to loss function optimization. Light GBM is selected as the predictive algorithm with AutoML using 10-second training times on the same label/feature data displayed in Table 2. The results of the loss function magnitudes are displayed in Figure 20.

Figure 20 shows four different plots for each loss function as designated on the y-axis. The uppermost left plot, for example, displays the resulting values of R^2 when AutoML is instructed to optimize for each different loss function along the x-axis (specified by text labels). This data shows that when R^2 is selected as the loss function to optimize, it also has the highest value of R^2 in the group (followed by MAE, RMSE, and MSE).

In the case of MSE and RMSE however, AutoML is able to optimize these loss metrics better when R^2 is selected rather than the optimized loss function itself (on the training data).

CONCLUSIONS

MAE is compiled by the type of algorithm used to predict PEF, broken down into the categories of decision tree, gradient descent, linear, DTA, and NN as depicted in Figure 21.

Decision-tree-based algorithms tend to perform with the highest predictive accuracy and lowest MAE followed by DTA, linear, NN, and gradient descent.

Gradient descent-based algorithms tend to perform very poorly matching log data with inconsistent repeatability when trained with AutoML. The Neural Network module also performed poorly, despite an array of configuration parameters used to train it.

In terms of training time, ease of use, and accuracy, the best algorithms found to predict PEF data in order are Light GBM, Fast Tree, Fast Tree Tweedie, Fast Forest, and GAM.

Figure 21: MAE by algorithm type

NOMENCLATURE

Abbreviations

API = application programming interface
CNN = convolutional neural network
CV = cross validation
DART = dropout meets multiple additive regression trees
DNN = deep neural network
DT = decision tree
DTA = domain transfer analysis
DTC = compressional sonic slowness
E = entropy
EE = experienced eye
ERR = error
FF = fast forest
FT = fast tree
FTT = fast tree tweedie
GAM = generalized additive model
GBDT = gradient-boosted decision trees
GBM = gradient-boosted machine
GD = gradient descent
GOSS = gradient one-sided sampling
GR = gamma ray
IG = information gain
IP = interactive petrophysics
L1 = LASSO regularization parameter
L2 = ridge regularization parameter
LASSO = least absolute shrinkage and selection operator
L-BFGS = limited memory Broyden Fletcher Goldfarb Shanno
MAE = mean absolute error
ML.Net = Microsoft machine learning .NET library
MLR = multiple linear regression
MSE = mean squared error
NN = neural network
NPHI = thermal neutron porosity
OGD = online gradient descent
OLS = ordinary least squares
PEF = photoelectric factor
PFI = permutation feature importance
R^2 = coefficient of determination
RDEEP = deep resistivity
RHOB = bulk density
RMSE = root mean squared error
SDCA = stochastic dual coordinate ascent

Symbols

y = function value
$\hat{y}$ = predicted or estimated value of function y
e = error or residual
λ = regularization parameter
γ = weighting parameter for child nodes

n = total number of samples in a set
i = iteration count, position in set
p = probability
w = model weight

REFERENCES

Banas, R., McDonald, A., Perkins, T., 2021, Novel Methodology for Automation of Bad Well Log Data Identification and Repair, SPWLA 62nd Annual Logging Symposium, May 17-20, 2021

Ciampiconi, L., Elwood, A., Leonardi, M., Mohamed, A., Rozza, A., A Survey and Taxonomy of Loss Functions in Machine Learning, 2023.

Dodge, Y., (2008) *The Concise Encyclopedia of Statistics*, Springer, New York, NY. ISBN 978-0-387-31742-7.

Guolin, K., Meng, Q., Finley, T., Wang, T., Chen, W., Weidong, M., Qiwei, Y., Tie-Yan, L., LightGBM: A Highly Efficient Gradient Boosting Decision Tree, 31st Conference on Neural Information Processing Systems (NIPS), 2017, Long Beach, CA.

Hastie, T., Tibshirani, R., Friedman, J., (2017), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, (Second Edition), Springer, New York, NY.

Hsiang-Fu, Y., Cho-Jui, H., Kai_Wei, C., Chih-Jen, L., Large Linear Classification When Data Cannot Fit in Memory, Knowledge Discovery from Data (KDD), July 25-28, 2010, Washington DC, USA

Lou, Y., Caruana, R., Gehrke, J, 2012, Intelligible Models for Classification and Regression, Knowledge Discovery and Data Mining (KDD), Beijing, China, 12-16 August.

Lou, Y., Caruana, R., Gehrke, J., Intelligible Models for Classification and Regression, Knowledge Discovery from Data (KDD), August 12-16, Beijing, China

Microsoft ML.Net API Documentation, (2024, October 1), from https://learn.microsoft.com/en-us/dotnet/machine-learning/

Microsoft Light GBM Winning Solutions (2024, October 1), from https://github.com/Microsoft/LightGBM/tree/master/examples

Rashmi, K.V., Gilad-Bachrach, R., DART: Dropouts meet Multiple Additive Regression Trees, Proceedings of 18th International Conference on Artificial Intelligence and Statistics, 2015, San Diego, CA, USA.

Shalev-Shwartz, S., Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization, Journal of Machine Learning Research 14, 2013, pp. 567-599

Soyoung, L., Simple Explanation of LightGBM without Complicated Mathematics (March 16, 2024), from https://medium.com/@soyoungluna/simple-explanation-of-lightgbm-without-complicated-mathematics-973998ec848f

ABOUT THE AUTHORS

Ryan Banas is a petrophysicist, petroleum engineer and licensed professional geoscientist with over 20 years of experience working for both service and operating companies. He is currently the managing director of PetroRes Consulting providing petrophysics, geoscience, engineering, and software consulting services to clients worldwide. Previously he has worked for Apache Corporation, Michigan Technological University, Schlumberger, Entergy Nuclear, and Precision Consulting