Machine Learning in Rock Facies Classification: An Application of XGBoost

Licheng Zhang, Cheng Zhan

Summary

Big data analysis has drawn much attention across different industries. Geoscientists, meanwhile, have been doing analysis with voluminous data for many years, without even bragging how big it is. In this paper, we present an application of machine learning, to be more specific, the gradient boosting method, in Rock Facies Classification based on certain geological features and constrains. Gradient boosting is a both popular and effective approach in classification, which produces a prediction model in an ensemble of weak models, typically decision trees. The key for gradient boosting to work successfully lies in introducing a customized objective function and tuning the parameters iteratively based on cross-validation. Our model achieves a rather high F1 score in evaluating two test wells data.

Introduction and Background

Machine learning emerges to be a very promising area and should make the work of future geoscientists more fun and less tedious. Furthermore, with the maturing neural network technology, the ability for better geological interpretation could be more automatic and accurate, e.g., in the Gulf of Mexico region, salt body characterization (challenging in the velocity model) might be elevated to the next level of higher quality seismic images.

There are a few decision tree based algorithms to handle classification problems. One is using the random forest, which operates by constructing multiple decision trees to reduce the possible variance error in each model. Another widely used technique is gradient boosting, which has been successfully applied in many Kaggle competitions. This method focuses on where the model performs poorly, and improves those areas by introducing a learner to compensate the existing model.

This facies classification problem was originally introduced in the Leading Edge by Brendon Hall in Oct. 2016 **(Hall, 2016)**. It seems to evolve into the first machine learning contest in the SEG, more information to be found on here (https://github.com/seg/2016-ml-contest). By the time we submitted the paper, our ranking is 5th on the leaderboard.

This data is from the Council Grove gas reservoir in Southwest Kansas. The Panoma Council Grove Field is predominantly a carbonate gas reservoir encompassing 2700 square miles in Southwestern Kansas. This dataset is from ten wells (with 4149 examples), consisting of a set of seven predictor variables and a rock facies (class) for each example vector and validation (test) data (830 examples from two wells) having the same seven predictor variables in the feature vector. Facies are based on the examination of cores from nine wells taken vertically at half-foot intervals. Predictor variables include five from the wireline log measurements and two geologic constraining variables that are derived from geologic knowledge. These are essentially continuous variables sampled at a half-foot sample rate.

The seven predictor variables are:

Five wireline log measurements	**Two geologic constrains**
● Gamma ray (GR) ● Resistivity logging (ILD_log10) ● Photoelectric effect (PE) ● Neutron-density porosity difference (Delta PHI) ● Average neutron-density porosity (PHIND)	● Nonmarine-marine indicator (NM_M) ● Relative position (RELPOS)

The nine discrete facies (classes of rocks), the abbreviated labels, and the corresponding adjacent facies are listed in the following Table 1. The facies gradually blend into one another, and some of the neighboring facies are rather close. Mislabeling within these neighboring is possible to occur.

Table 1:

Class of rocks	**Facies**	**Label**	**Adjacent Facies**
Nonmarine sandstone	1	SS	2
Nonmarine coarse siltstone	2	CSiS	1,3
Nonmarine fine siltstone	3	FSiS	2
Marine siltstone and shale	4	SiSh	5
Mudstone	5	MS	4,6
Wackestone	6	WS	5,7
Dolomite	7	D	6,8
Packstone-Grainstone	8	PS	6,7,9
Phylloid-glgal bafflestone	9	BS	7,8

Methodology

Generally speaking, there are 3 types of machine learning algorithms: supervised learning, unsupervised learning, and reinforcement learning. The application in this paper belongs to the category of supervised learning. This type of algorithm consists of a target/outcome variable (or dependent variable), which is to be predicted from a given set of predictors (independent variables, or usually called features). Using these feature variables, a function that maps inputs to desired outputs will be generated. The training process continues until the model achieves a satisfied level of accuracy on the training data. Examples of supervised learning includes: regression, decision tree, random forest, KNN, logistic regression etc.

The algorithm adopted here is called XGBoost (eXtreme Gradient Boosting), which is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way. It was created and developed by Tianqi Chen, a Ph.D. student at the University of Washington. More details about XGBoost can be found here (http://dmlc.cs.washington.edu/xgboost.html).

The basic idea of boosting is to combine hundreds of simple trees with low accuracy to build a more accurate model. Every iteration will generate a new tree for the model. When it comes to how a new tree is created, there are thousands of methods. A famous one is called Gradient Boosting machine, raised by Friedman **(Friedman, 2001)**. It utilizes the gradient descent to generate the new tree based on all previous trees, driving the objective function towards the minimum direction.

An objective function usually has the form that contains two parts (training loss and regularization):

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta) \tag{1}$$

Where $\boldsymbol{L}$ is the training loss function, and Ω is the regularization term. The training loss measures the performance of the model is on training data. The regularization term controls the complexity of the model, which usually controls overfitting. The complexity of each tree is defined as the following:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 \tag{2}$$

There is, of course, more than one way to define the complexity, and this particular one works well in practice. And the objective function in XGBoost is defined as:

$$obj = \sum_{j=1}^{T}[G_j\omega_j + \frac{1}{2}(H_j + \lambda)\omega_j^2] + \gamma T \tag{3}$$

More details about the notations can be found here (http://xgboost.readthedocs.io/en/latest/model.html).

Data Analysis and Model Selection

Before building any machine learning model, it is necessary to perform some exploratory analysis and cleanup.

First, we examine the data that will be used to train the classifier. The data consists of 5 wireline log measures, 2 indicator variables, and 1 facies label at half foot interval. In machine learning terminology, each log measurement is a feature vector that maps a set of 'features' (the log measures) to a class (the facies type).

Pandas library in Python is a great tool in loading data into the dataframe structure for further manipulation.

```
In [2]: import pandas as pd
        filename = './facies_vectors.csv'
        training_data = pd.read_csv(filename)
        training_data.head(10)
```

Out[2]:

	Facies	Formation	Well Name	Depth	GR	ILD_log10	DeltaPHI	PHIND	PE	NM_M	RELPOS
0	3	A1 SH	SHRIMPLIN	2793.0	77.45	0.664	9.9	11.915	4.6	1	1.000
1	3	A1 SH	SHRIMPLIN	2793.5	78.26	0.661	14.2	12.565	4.1	1	0.979
2	3	A1 SH	SHRIMPLIN	2794.0	79.05	0.658	14.8	13.050	3.6	1	0.957
3	3	A1 SH	SHRIMPLIN	2794.5	86.10	0.655	13.9	13.115	3.5	1	0.936
4	3	A1 SH	SHRIMPLIN	2795.0	74.58	0.647	13.5	13.300	3.4	1	0.915
5	3	A1 SH	SHRIMPLIN	2795.5	73.97	0.636	14.0	13.385	3.6	1	0.894
6	3	A1 SH	SHRIMPLIN	2796.0	73.72	0.630	15.6	13.930	3.7	1	0.872
7	3	A1 SH	SHRIMPLIN	2796.5	75.65	0.625	16.5	13.920	3.5	1	0.830
8	3	A1 SH	SHRIMPLIN	2797.0	73.79	0.624	16.2	13.980	3.4	1	0.809
9	3	A1 SH	SHRIMPLIN	2797.5	76.89	0.615	16.9	14.220	3.5	1	0.787

Then some basic statistical analysis are produced, for example, the distribution of each classes (Figure 1a), heatmap of features (Figure 1b), which produces correlation plot for us to observe relationship between variables, and log plots for wells (Figure 1c). These figures are the initial blocks to explore the data, and the visualization libraries are seaborn and matplotlib.

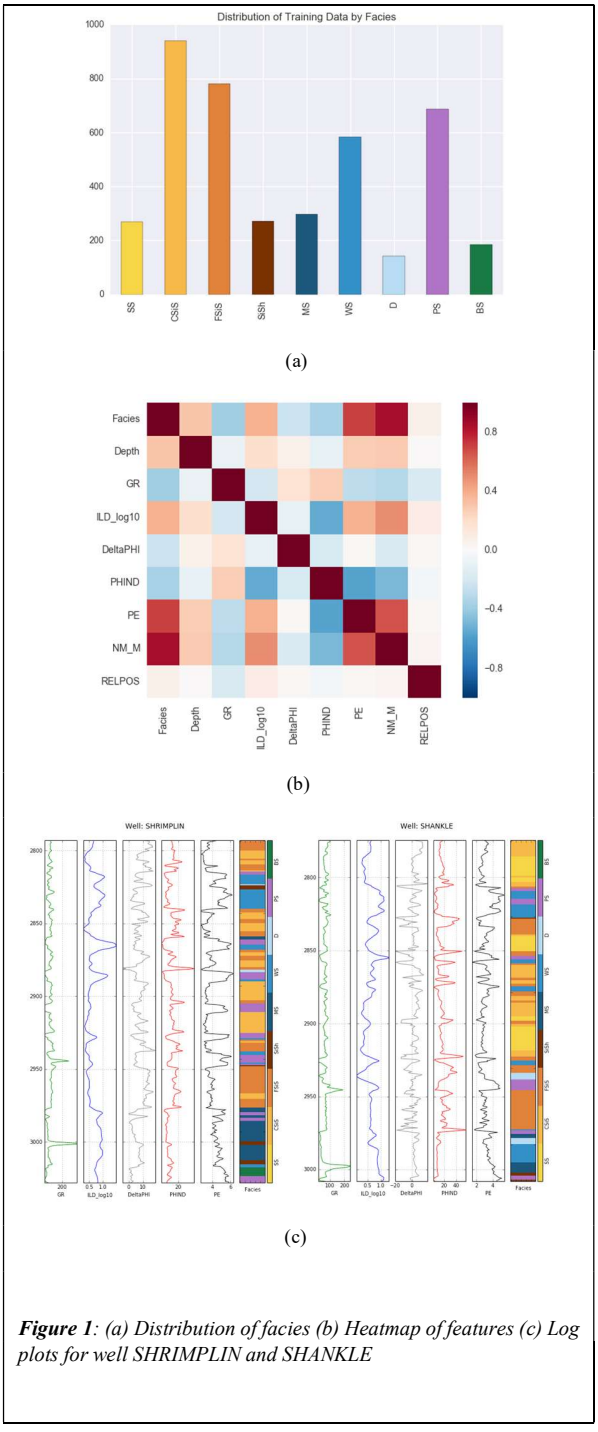


***Figure 1**: (a) Distribution of facies (b) Heatmap of features (c) Log plots for well SHRIMPLIN and SHANKLE*

The next step is data preparation and model selection. The goal is to build a reliable model to predict the Y values (Facies) based on X values (the seven predictor variables).

To enhance the performance of XGBoost's speed over many iterations, we create a DMatrix format. Such process sorts the data initially to optimize for XGBoost in building trees, and reduces the runtime correspondingly. This is especially helpful in learning with a large number of training examples.

```
import xgboost as xgb
X_train = training_data.drop(['Facies', 'Well Name','Formation','Depth'], axis = 1 )
Y_train = training_data['Facies' ] - 1
dtrain = xgb.DMatrix(X_train, Y_train)
```

On the other hand, in order to quantify the quality of the models, certain metrics are needed. We use accuracy metrics for judging the models. A simple and easy way to learn the terminologies (e.g., accuracy, prediction, recall) can be found in the following webpage (http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/).

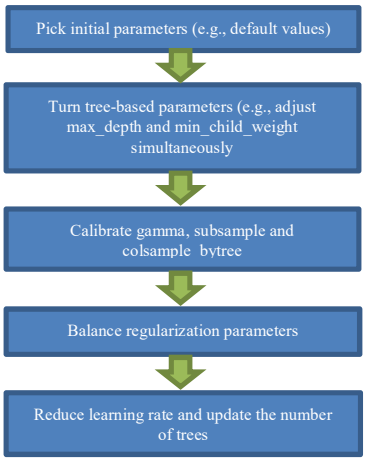
There are several main parameters to be tuned to get a good model for this rock facies classification problem.

```
from xgboost import XGBClassifier
xgb1 = XGBClassifier(learning_rate = 0.1, n_estimators=1000,
                     max_depth=5, min_child_weight=1, gamma = 0, subsample=0.8,
                     colsample_bytree=0.8, objective='multi:softmax', nthread =4
)
```

Table 2: main parameters

Learning rate	Step size shrinkage employed to prevent overfitting. We shrinks the feature weights to make the boosting process more conservative
N_estimators	The number of trees
Max_depth	Maximum depth of a tree, and increasing this value will make the model more complex(likely to be overfitting)
Min_child_weight	Minimum sum of instance weight needed in a child
Gamma	Minimum loss reduced required to make a further partition on a leaf node of the tree
Subsample	Subsample ratio of the training instance
Colsample_bytree	Subsample ratio of features when constructing each tree
Objective:'multi:softmax'	This sets XGBoost to produce multiclass classification using the softmax objective
nthread	Number of parallel threads used to run XGBoost

Algorithm parameter tuning is a critical processs in achieving the optimal performance of certain algorithm, and needs to be carefully justified before moving into production. Our workflow for optimizing parameters is presented here:



The reason we adopt such flow is because of the nature of XGBoost algorithm, which is robust enough not to be overfitting with increasing trees, but a high value for a particular learning rate could degrade its ability in predicting new test data. As we reduce the learning rate and increase the number of trees, the computation becomes expensive and could potentially take longer time on standard personal computers.

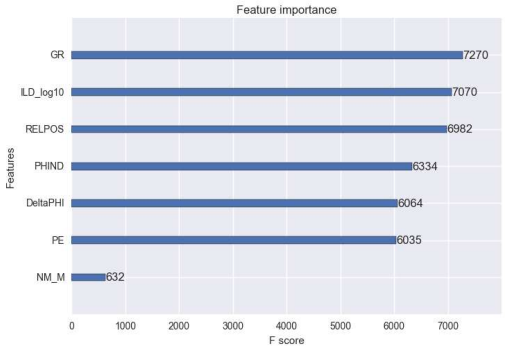
Grid search is a typical approach for parameters tuning that methodically builds and evaluates a model for each combination of parameters in a specific grid. For instance, the code below examines different combinations of 'max_depth' and 'min_child_weight'.

```
from sklearn.model_selection import GridSearchCV
param_test1={
    'max_depth':range(3,10,2),
    'min_child_weight':range(1,6,2)
}

gs1 = GridSearchCV(xgb1,param_grid=param_test1,
                   scoring='accuracy', n_jobs=4,iid=False, cv=5)
gs1.fit(train[features],train[target])
gs1.grid_scores_, gs1.best_params_,gs1.best_score_
```

Another way to tailor parameters is by random search, which complements the predefined grid search procedure that is currently being exploited. In this case, we didn't find random search benefits much the final results.

After several iterations, the final model is built up. A cross-validation is conducted to access the performance before applying to another two blind well test data. The best accuracy (F1 score) we have so far is 0.564, ranked 5th in the contest. The following is the feature importance plot of the model. Importance provides a score that indicates how useful or valuable each feature was in the construction of the boosted decision trees within the model. The more an attribute is used to make key decisions with decision trees, the higher its relative importance.



Conclusions

We have successfully applied the gradient boosting method to a classification problem in the rock facies. Potential applications of such prediction could be to validate the velocity model for seismic data. This could be viewed as some commencing endeavors for more machine learning applications in the near future of the oil and gas sector.

Acknowledgments

The authors would like to thank Ted Petrou, Aiqun Huang and Zhongyang Dong for discussion. We also thank Yan Xu for reviewing the manuscript.

Reference

Chen, T. & Guestrin, C., 2016. Xgboost: A scalable tree boosting system. *arXiv preprint arXiv:1603.02754.*

Friedman, J. H., 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189-1232.

Hall, B., 2016. Facies classification using machine learning. *The Leading Edge*, 10, pp. 906-909.

Natekin, A. & Knoll, A., n.d. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics,* p. 2013.