Pseudosonic Log Generation With Machine Learning A Tutorial for the 2020 SPWLA PDDA SIG ML Contest

Yanxiang Yu, Siddharth Misra, Osogba Oghenekaro, and Chicheng Xu

ABSTRACT

Compressional and shear sonic traveltime logs (DTC and DTS respectively) are crucial for subsurface geomechanical characterization and seismic-well tie. However, these two logs are often missing or incomplete in many oil and gas wells. In this tutorial, we applied the machine-learning algorithm and used seven "easy-to-acquire" conventional logs to predict the DTC and DTS logs. A total number of 20,525 data points (corresponding to distinct depths) collected from three wells were used to train regression models using machine-learning techniques. Each of the data points has seven features, which are the conventional "easy-to-acquire" logs, namely caliper, neutron porosity, gamma ray, deep resistivity, medium resistivity, photoelectric factor and bulk density, respectively, and two targets, which are the sonic traveltime logs, DTC and DTS, respectively. The objective is to develop regression models that can process the seven features and generate the two targets. Various data preprocessing and supervised-learning techniques from Scikit-learn toolbox are applied to train the regression models. Random forest (RF) regressor has the best performance in synthesizing DTC and DTS logs at R^2 score of 0.988. Finally, for purposes of blind test, the RF regressor is applied on the hidden dataset from a different well. The root-mean-square-error (RMSE) value achieved in the blind test is provided to the competition organizers for ranking our performance relative to other participants.

INTRODUCTION

Sonic traveltime logs contain critical geomechanical information for subsurface characterization around the wellbore. Often, the sonic logs are required to complete the well-seismic tie workflow or geomechanical properties prediction. However, due to budget control and operational issues, these logs are not always acquired and are run in limited number of wells. Other well logs, like gamma ray, resistivity, density, and neutron logs, are considered "easy-to-acquire" conventional well logs that are run in most of the wells. When sonic logs are absent in a well or an interval, a common practice is to synthesize them based on neighboring wells that have sonic logs and their subsurface properties from the conventional logs. See He et al. (2018, 2019) for additional details.

There has been an increasing excitement about applying machine-learning and artificial intelligence (AI) methods in the oil and gas industry. In this approach, the sonic log synthesis or prediction by processing conventional logs using machine-learning techniques is a perfect demonstration of the power of machine-learning application. Many free and open-source packages now exist that provide powerful additions to the petrophysists' or rock physicist's' toolbox. One of the best examples is scikit-learn (http://scikit-learn.org/), a collection of tools for machine learning in Python to compete the-machine learning process for this problem. Check Hall (2016) for more details. In this tutorial, we'll be using functions from this library and provide a machine-learning workflow to predict the DTC and DTS logs by processing conventional logs. The prediction models are trained by processing data from Well 1 data, and use feature sets derived from the seven conventional logs: caliper, neutron, gamma ray, deep resistivity, medium resistivity, photoelectric factor and density,. Then the model is used to generate the two targets, i.e., DTC and DTS logs, in a similar Well 2. The predicted values are saved in the same format as the given sample_submission.csv, and submitted together with notebook for judgement.

EXPLORING THE DATASET

The dataset we use comes from the Equinor Volve field data from the link [https://www.equinor.com/en/how-and-why/digitalisation-in-our-dna/volve-field-data-village-download.html]. We use Pandas library to load the data into a dataframe, which provides a convenient data structure to visualize and perform exploratory data analysis on the available logging data. For example, we apply the data.describe() function to gain a quick overview of the statistical distribution of the training data, as shown in Table 1.

```
>>> import pandas as pd
>>> df1 = pd.read_csv('train.csv')
>>> df1.describe()
```

Table 1–Statistical Distribution of the Original Training Dataset

	CAL	CNC	GR	HRD	HRM	PE	ZDEN	DTC	DTS
count	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000	30143.000000
mean	-8.394576	-23.692615	38.959845	3.977690	1.547299	-17.446739	-20.229982	-54.891703	-9.737567
std	129.970219	157.142679	108.504554	365.112753	456.908969	149.083136	148.809506	372.858812	440.314119
min	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000	-999.000000
25%	8.058350	0.122800	17.248750	0.717700	0.712050	0.053100	2.226700	66.304350	118.534350
50%	8.625000	0.193600	36.821800	1.623000	1.628100	4.941500	2.432200	78.355100	137.689300
75%	9.048850	0.337150	58.346150	3.158300	3.280600	7.856650	2.551350	107.022500	182.973150
max	21.064200	3490.158200	1470.253400	10000.000000	60467.761700	28.106400	3.259700	155.980300	487.438400

From Table1, we can see a total of 30,413 samples (data vectors) are loaded, and each of them consists of nine data columns: caliper (CAL), neutron (CNC), gamma ray (GR), deep resistivity (HRD), medium resistivity (HRM), photoelectric factor (PE), density (ZDEN), compressional traveltime (DTC) and shear traveltime logs (DTS).

Handling the Missing Data

"-999" is shown as the missing values in all features. To handle the missing values, we first replace all the values equal to "-999" to "np.nan", and then remove all the rows that contain the nan by using data.dropna() function. This is another quick implementation provided in the Pandas dataframe. After removing all the missing values, there are 20,525 data vectors left as shown in Table 2.

Identifying the Features and Targets

In this dataset, the first seven data columns are the features that are required for the desired machine-learning task, and the last two data columns are the targets. We extract the feature vectors and the associated target vectors from the training and testing dataset as:

```
>>> y_target = df1_data[:,-2:]
>>> X_feature = df1_data[:,:-2]
```

Splitting the Dataset into Training and Testing Datasets

A standard practice before doing any further data preprocessing and training the supervised-learning model is to separate the data into the training and testing datasets, where the testing set can be used to evaluate the generalization of the model in terms of overfitting or underfitting. Once the train-test split is performed, the test dataset should not be touched, to avoid information leakage from testing dataset to training dataset. The testing dataset should be used only for purposes of evaluation the generalization capability of the model. More information on this can be found in Misra et al. (2019b). In the code shown below, we randomly separate the training dataset to 80% training set and 20% testing set. There are several other ways of splitting the dataset as shown in the references.

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X_feature, y_target, test_size=0.2)
```

Outlier Detection

One of the findings from Table 2 is that the maximum values of all features are dramatically larger than their mean values, which indicates anomalies and outliers exist in the dataset. Therefore, some special treatments may be helpful to improve the performance of the model trained. Here, we haven't explored any other methods other than removing the missing values. We suggest that the contestants try their best to quality control the log data. More information on this can be found in Misra et al. (2019a).

Data Transformation Using Scalers

While many machine-learning algorithms assume the feature data to be normally distributed with zeros mean and unit variance, from Table 2, we can see it's clearly not the case with our training data. StandardScaler from sklearn.preprocessing toolbox is a handy function that can help to standardize the input data, and the following codes shows the standardization process. It is important to note that scaling should be performed first on the training dataset to learn the scaling parameters. Following that, the entire testing dataset should be transformed using the scaler that learned the scaling parameters from the training dataset. When the entire dataset is scaled at the same time, it will lead to data leakage

Table 2—Statistical Distribution of the Training Dataset After Removing Missing Values

	CAL	CNC	GR	HRD	HRM	PE	ZDEN	DTC	DTS
count	20525.000000	20525.000000	20525.000000	20525.000000	20525.000000	20525.000000	20525.000000	20525.000000	20525.000000
mean	8.426679	0.274416	49.889253	2.598719	5.835466	3.833792	2.410734	88.312221	182.051067
std	1.845912	3.062495	54.811017	3.465665	422.449589	4.375818	0.181713	23.542419	84.670122
min	5.930400	0.014500	1.038900	0.123600	0.134100	-0.023200	0.680600	49.970500	80.580400
25%	6.629100	0.120300	16.036800	0.810000	0.797300	0.049800	2.236100	70.423100	127.148800
50%	8.578100	0.187700	37.498000	1.814900	1.829300	3.287800	2.466500	79.695400	142.678500
75%	8.671900	0.329000	61.140700	3.337400	3.463300	7.061300	2.563700	102.482800	192.757800
max	21.064200	365.885000	1470.253400	206.718200	60467.761700	28.106400	3.259700	155.980300	487.438400

between the training and testing dataset. More information on this can be found in Misra and He (2019).

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train_scaled = scaler.fit_transform(X_train)
>>> X_test_scaled = scaler.transform(X_test)
```

Handling missing values, train-test split, removing outliers, and data transformation using scalers and normalizers are standard steps involved in the data preprocessing before building the machine-learning model.

Training the Supervised Model

We now have the data ready for training a supervised regression model. Sklearn library provides many convenient functions for the regression model, LinearRegression is a good starting point that acts as baseline. Another baseline that is nonlinear in nature is provided by k-nearest neighbor regressor. The GridSearchCV method from sklearn.model_selection function needs to be used to ensure that the regression method is trained and evaluated on all the statistical variations in the training dataset so that we can find the most generalizable form of the regressor. GridSearchCV should be performed only on the training dataset. GridsearchCV performs hyperparameter optimization of *n* hyperparameters for each split out of the *k* total splits of the training dataset. If there are n hyperparameters and k-fold cross-validation is desired, then k*n models will be trained and evaluated on the various splits of the training dataset. R-squared (R^2) score is used as the scoring criteria to evaluate the best model. The code below shows few of the steps in training the RandomForestRegressor.

```
>>> from sklearn.model_selection import GridSearchCV
>>> clf = RandomForestRegressor(n_estimators=100)
>>> grid = GridSearchCV(estimator=clf, param_grid=param_
grid, scoring='r2', cv=5)
>>> grid.fit(X_train_scaled, y_train)
>>> best_model = grid.best_estimator_
```

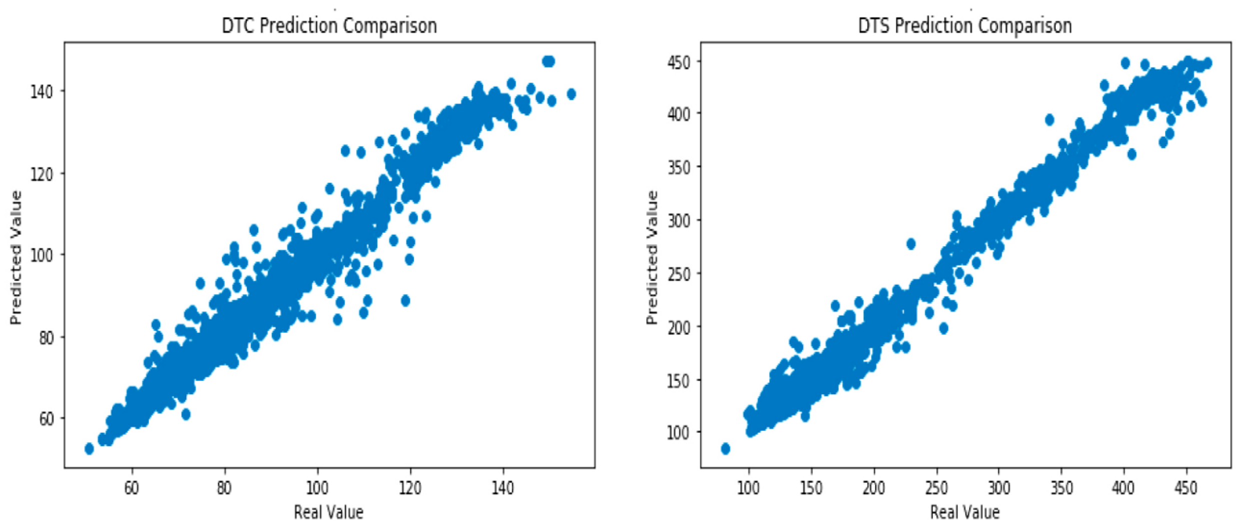
After training on the X_train and y_train, the random forest regression model needs to be evaluated on the test dataset. The random forest regressor exhibits a good performance on the test dataset, which in terms of R^2 is 0.988 and RMSE is 5.55. Figure 1 shows the predicted value versus the original value for the testing dataset, we can see a very good match.

Blind Testing on the Hidden Dataset

The random forest regressor is then applied to the hidden dataset for purposes of blind testing. Note that the blind-test data also need to be transformed with the same scaler generated by train dataset. After all the values are predicted, we'll save it to a csv file, and submit it to the committee for scoring, as shown below:

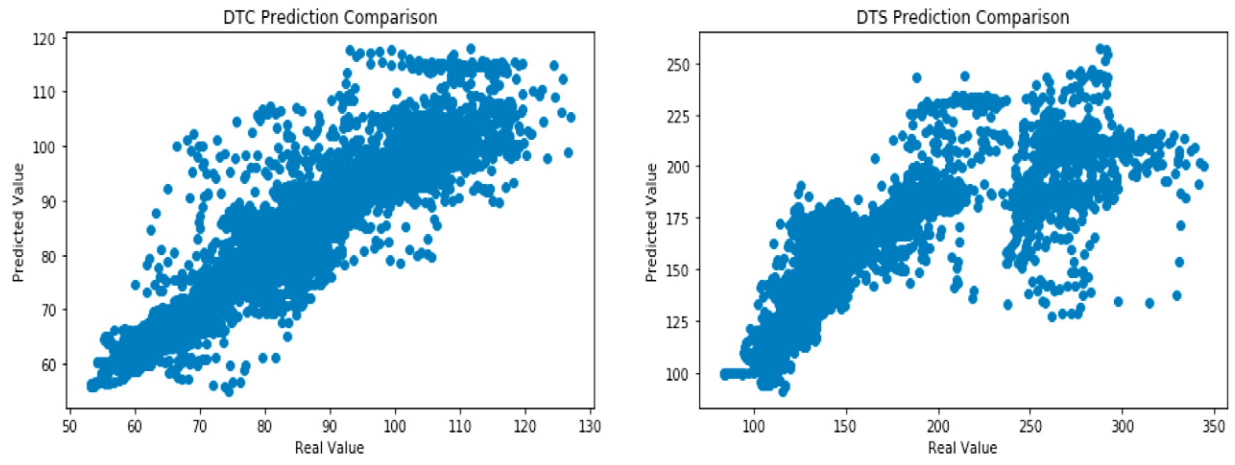
```
>>> df2 = pd.read_csv('test.csv')
>>> for col in df2.columns.tolist():
>>>     df2[col][df2[col]==-999] = np.nan
>>> df2.dropna(axis=0, inplace=True)
>>> df2_data = np.array(df2)
>>> x_trainwell2 = scaler.transform(df2_data)
>>> well2_predict = RF_best.predict(x_trainwell2)
>>> output_result = pd.DataFrame({'DTC':well2_predict[:,0],
'DTS':well2_predict[:,1]})
>>>       output_result.to_csv(path_or_buf='./submission.csv',
index=False)
```

The comparisons between the predicted results with the true values in the hidden test dataset are shown in Fig. 2.



(a) (b)

Fig. 1—Predicted values of (a) DTC, and (b) DTS versus the true values on the testing dataset from the random forest model.



(a) (b)

Fig. 2—Predicted values of (a) DTC, and (b) DTS versus true values in the hidden test dataset

The RMSE score of 17.79 is achieved, and it is over three times larger than the performance on the test dataset. From Fig. 2, we can clearly see the DTS prediction is less accurate than the DTC prediction, especially in the zones of large DTS (slow formations). Therefore, it is suggested the pseudolog prediction should be limited to a certain stratigraphic interval instead of the whole depth.

CONCLUSION

In this tutorial, we demonstrate using a machine-learning workflow on a practical petrophysical problem: preparing a dataset, training and testing a regression model, and finally blind-testing (similar to the real-world deployment) the model on the hidden dataset. Libraries and open-source tools, such as scikit-learn provide powerful algorithms that can be applied to problems with few lines of code, which greatly helps to facilitate the research of data science in the petrophysics area.

In addition to the procedures mentioned above, many other methods may be applied to improve the performance and stability of the model, such as making special treatments to the anomalies and outliers, train different models for zones that shows very distinct DTC/DTS range, training other regression models and/or combining them.

For more details about the data and code, please check the Github repo: https://github.com/pddasig/Machine-

Learning-Competition-2020.

ACKNOWLEDGEMENTS

A note of thanks goes to Equinor for releasing the Volve dataset. We also thank the members of the SPWLA PDDA SIG ML Contest Committee Brendon Hall, Yan Xu, Michael Ashby and Weichang Li for their contributions.

REFERENCES

Hall, B., 2016, Facies Classification Using Machine Learning, *The Leading Edge*, **35**(10), 906–909. DOI: 10.1190/tle35100906.1.

He, J., Misra, S., and Li, H., 2018, Comparative Study of Shallow Learning Models for Generating Compressional and Share Traveltime Logs, *Petrophysics*, **59**(6), 826–840. DOI: 10.30632/PJV59N6-2019a7.

He, J., Li, H., and Misra, S., 2019, Data-Driven In-Situ Sonic-Log Synthesis in Shale Reservoirs for Geomechanical Characterization, Paper SPE-191400, *SPE Reservoir Evaluation & Engineering*, **22**(4), 1225–1239. DOI: 10.2118/191400-PA.

Misra, S., Chakravarty, A., Bhoumick, P., and Rai, C.S., 2019b, Unsupervised Clustering Methods for Noninvasive Characterization of Fracture-Induced Geomechanical Alterations, Chapter 2, *in Machine Learning for Subsurface Characterization*, Elsevier, 39–65. ISBN: 978-0-12-817736-5.

Misra, S., and He, J., 2019, Stacked Neural Network Architecture to Model the Multifrequency Conductivity/Permittivity Responses of Subsurface Shale Formations, Chapter 4, *in Machine Learning for Subsurface Characterization*, Elsevier, 103–127. DOI: 10.1016/B978-0-12-817736-5-00004-1.

Misra, S., Osogba, O., and Powers, M., 2019a, Unsupervised Outlier Detection Techniques for Well Logs and Geophysical Data, Chapter 1, *in Machine Learning for Subsurface Characterization*, Elsevier, 1–37. ISBN: 978-0-12-817736-5.