MACHINE LEARNING
APPLIED TO MULTIPHASE PRODUCTION PROBLEMS

A THESIS
SUBMITTED TO THE PROGRAM IN ENERGY RESOURCES ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Tita Ristanto
June 2018

I certify that I have read this thesis and that, in my opinion, it is fully adequate in scope and quality, as partial fulfillment of the degree of Master of Science in Energy Resources Engineering.

(Prof. Roland N. Horne) Principal Adviser

Abstract

In reservoir engineering, it is important to understand the behavior of the reservoir, often by seeing the dynamics of bottom hole pressure (p_{wf}) and flow rate (q) in each well. Bottom hole pressure and flow rate are commonly modeled using physical model, for example, reservoir numerical simulator. This approach requires physical parameters, including rock properties and reservoir geometry, which we do not always know. In some cases, collecting those physical parameters and building the physical model might not be practical. Applying this approach in a field that has thousands of wells with a very dynamic environment requires a lot of effort.

Another approach uses historical bottom hole pressure and flow rate, which are commonly easier to obtain. Using machine learning algorithms, we can build a reservoir model by learning the historical pattern of bottom hole pressure and flow rate (or training set) without the physics of the flow being programmed explicitly. Using this model, we can predict flow rate given bottom hole pressure (or vice versa). We can also utilize this model as a diagnostic tool. For example, if liquid loading or condensate banking or wax/asphaltene deposition starts to happen, the actual pressure and flow rate response will be different than that suggested to the model, hence flagging that the reservoir performance has changed in character.

This research focused on solving multiwell and multiphase problems using machine/deep learning approaches. The study has identified important features in such problems. This study explored ten different machine/deep learning algorithms and their performance evaluation. Python is very helpful for this purpose as it has a lot of machine/deep learning libraries to choose from and is handier in solving machine learning related problems. The study also examined the impact of multiphase flow, reservoir heterogeneity, and data noise.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my adviser, Prof. Roland N. Horne, for the patient guidance and support through the years of my Master's degree. I am truly inspired by his spirit and dedication in academics.

I have been blessed with friendly fellow students. I'd like to thank all my friends in Energy Resources Engineering Department and those in SUPRI-D group: Jason Hu, Dante Orta, Chuan Tian, Xuhua Gao, and Abdullah Alakeely, for their support, welcoming attitude, and knowledge sharing. I also would like to acknowledge the financial support from SUPRI-D Consortium for making my study and research work possible.

I cannot end the chain of gratitude without thanking my mother Asiyah for her continuous love and support. My special acknowledgement goes to my father, Subowo, and two brothers, Mulat Susilanto and Manon Pramono, for powerful lessons and values they instilled in me. I believe that they see me from the heaven and are proud of who I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Artificial Intelligence (AI) is transforming the oil and gas industry. A study from McKinsey (2016) in 2016 mentioned that digital organization, which covers AI and Machine Learning, is one of the five big ideas that will reshape the future of the industry. Digitizing and automating work will result in better safety and productivity as fewer people will be at risk and the risk of human error is reduced. Moreover, in the low oil price environment, oil companies often have no choice but to cut their operational budget, reduce the headcount, and/or fully optimize their remaining assets. This further encourages the adoption of artificial intelligence in the industry, with the hope to simplify and/or automate inefficient processes.

Current well production operation practices still have many challenges to solve. Not all wells have the luxury of having complete measurement devices. For instance, installing a multiphase flow meter in every well may not be economically feasible as these devices are expensive and the oil production might be too low to cover the cost. This is especially true in marginal wells. In high risk wells, for safety reasons, continuous real-time well measurement with small time interval is mandatory as missing one of the parameters in a certain period of time might lead to a disastrous effect. Ironically, in many cases where production data is present, the data are underutilized.

Typically, the flow rate data are used for forecast using Decline Curve analysis. However, such analysis does not take into account future bottom hole pressure fluctuation. Another commonly performed analysis is Inflow Performance Relationship (IPR) and Tubing Performance Relationship (TPR), but these methods are normally on the well-by-well basis and do not appear useful when it comes to time-series forecasting.

In the spirit of bringing AI into well production related matters, this research investigated some machine/deep learning model formulations not only to address multiphase production problems from multiple wells, but also unify them into a single model. Potential future implementation of the model includes automated production forecast given historical data. This might potentially reduce well surveillance and wellwork operations (to avoid obtaining the data that the model has already

understood).

There have been some implementations of AI in the field of oil and gas production. Boomer (1995) used Neural Network to forecast oil production in Vacuum Field, Texas and the model outperformed the professionals 93% of the time. Cao et al. (2016) found that Neural Network performed better than conventional decline curve methods (Arps, Duong, and Stretched Exponential Production Decline). Suhag et al. (2017) implemented Neural Network to predict oil rate in Bakken shale wells and yielded better results than Duong's method.

Previous efforts by Liu (2013) and Tian (2014), have been successful in applying machine learning method in single-phase production cases. Important features were identified and gave satisfactory results in several different cases. The study found that although pressure and flow rate relationships can be nonlinear, the problem can be formulated as a linear problem and the nonlinearity is expressed in the features. The objective of the current study was to go beyond single-phase cases to investigate machine learning applications in multiphase production.

This thesis proceeds as follows. Chapter 2 shows the process of building a machine/deep learning model and the basics of the machine/deep learning algorithms covered in this study. The model implementation is broken down to two different parts: 1) Bottom hole prediction, which will be covered in Chapter 3 and 2) Oil rate and water cut prediction, covered in Chapter 4. Finally, Chapter 5 concludes the study and provides some possible ideas for future work.

Chapter 2

Methodologies

This chapter describes the workflow of the research and the machine/deep learning algorithms used in this study. The first subsection elaborates the general workflow of this research. This chapter also explains how the dataset was generated, preprocessed, and trained and tested.

2.1 General Workflow

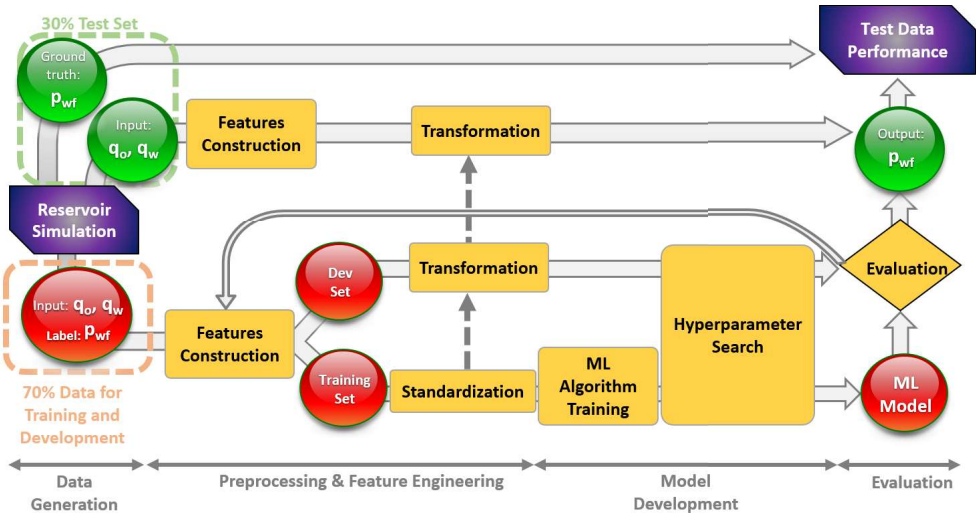

Figure 2.1: Workflow of building a machine/deep learning model in this study.

Figure 2.1 shows the entire process used to build the machine/deep learning models in this research. This is an example where the model takes flow rates as the input and bottom hole pressure as the output.

The flow rate and pressure dataset were generated using the ECLIPSE reservoir simulator. The data were split into training and development sets, together accounting for 70% of the total data. The remaining 30% was used as the test set. Feature construction transformed the raw input into meaningful forms, adding the nonlinearity and introducing the physics of the flow into the machine learning model. We further split 70% of the data into training (75%) and development (25%), see Fig. 2.2. Before feeding the features into the algorithm, we performed data standardization, which has been proven to be useful in various cases to speed up the optimization process.

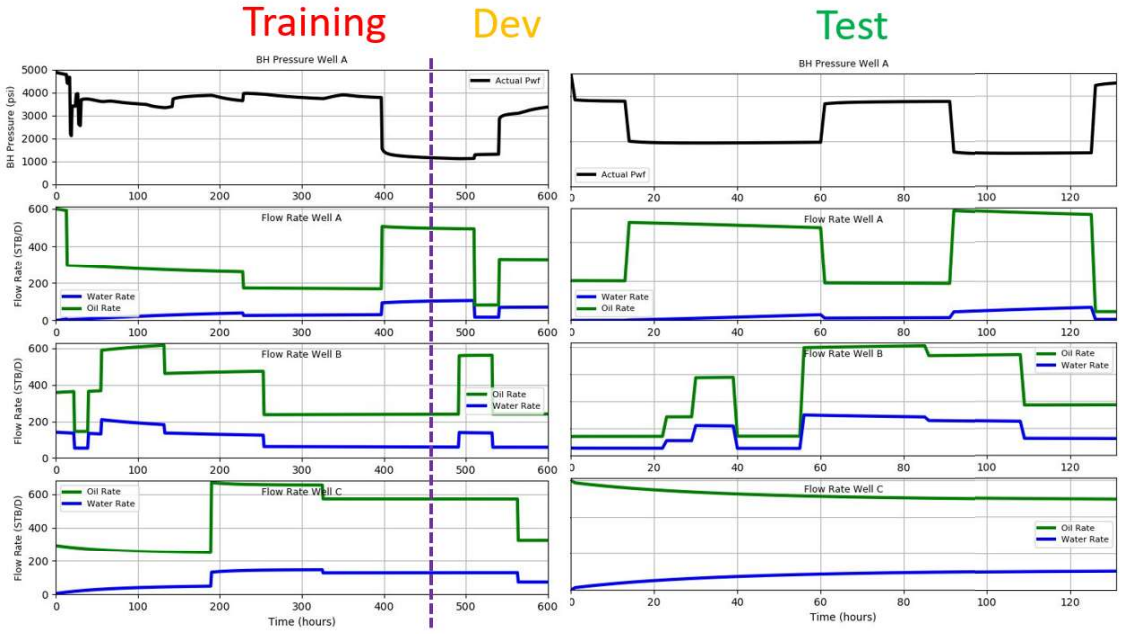

Figure 2.2: Visualization of training, development, and test sets.

The next step is the main part, where we trained ten different machine/deep learning algorithms. In each algorithm, we performed hyperparameter search to make sure that the resulting model has a set of parameters that maximize the dev set score, among all possible hyperparameter combinations. After tuning the model and obtaining the best one, we deployed the model in the test set and obtained the test score.

2.2 Dataset Generation and Preprocessing

2.2.1 Dataset Generation

A three-dimensional reservoir model was built using Eclipse with black-oil fluid formulation. The geometry and reservoir properties are described in Table 2.2.1. The reservoir model contains three

vertical production wells producing oil and water. The production is liquid rate-constrained. The simulation outputs oil rate, water rate, water cut, and bottom hole pressure from each well. This research assumed homogeneous reservoir properties and volumetric reservoir (no water-drive). Bubble point pressure was kept low to make sure that no free gas is produced from the reservoir.

Table 2.1: Reservoir, Rock, and Fluid Properties

Properties	Value	Unit
Reservoir length (L_x)	20000	ft
Reservoir width (L_y)	20000	ft
Reservoir height (L_z)	30	ft
Number of grids in x-direction	100	
Number of grids in y-direction	100	
Number of grids in z-direction	6	
Top layer depth (k_x)	5000	ft
Water-oil contact (k_x)	5025	ft
Permeability x-axis (k_x)	80	mD
Permeability y-axis (k_y)	80	mD
Permeability z-axis (k_y)	8	mD
Porosity (k_y)	0.2	(fraction)
Rock compressibility (c_r)	10^{-5}	psi^{-1}
Initial pressure (P_r)	5000	psi
Bubble point pressure (p_b)	0	psi

Table 2.2: Well Properties

Properties	Value	Unit
Well A completion interval	5000-5010	ft
Well B completion interval	5010-5025	ft
Well C completion interval	5005-5015	ft
Well diameter (for Well A, B, and C)	8.5	inches
Skin factor (for Well A, B, and C)	0	

2.2.2 Feature Engineering

In this study, five different feature formulations were used. The first and simplest feature formulation contains only raw input (without further modification). In the case of bottom hole pressure prediction, the inputs are oil and water rate formulated as follows.

$$X^{(i)} = \begin{bmatrix} q_o^{(i)} \\ q_w^{(i)} \end{bmatrix} \tag{2.1}$$

$$y^{(i)} = p_{wf}^{(i)}, \text{ where } i \text{ denotes the time step.} \tag{2.2}$$

Liu (2013) proposed a way to capture different flow behavior in the input features.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)/\left(t^{(i)} - t^{(j)}\right) \end{bmatrix}, \text{ where } j \text{ denotes the previous flow rate change events.} \tag{2.3}$$

We used Eq. 2.3 as our second feature formulation. The first feature captures the pressure response as a result of previous flow rate change events. The second feature gives the flavor of infinite-acting radial flow behavior. The third features describes wellbore storage and boundary effect. The fourth feature does not have any physical meaning but it has been shown to improve model performance. These four features perform well in a single well and single-phase problem. Tian (2015) continued the study and found out that these features also work well in multiwell and single-phase problems.

As this research focused primarily on oil and water (multiphase) production, we introduced a third feature formulation, in which we used these four features to represent the oil phase and four additional features to represent the water phase.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)/\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right)/\left(t^{(i)} - t^{(j)}\right) \end{bmatrix} \tag{2.4}$$

Solving multiwell problems requires an understanding of the Ei-function, which is an approximation to the full solution to the diffusivity equation in the case of interference, which has both logarithmic and exponential 'flavors'. Eq. 2.5 shows a simplified bottom hole pressure equation in well A as a function of flow rate and time of another well at distance r feet away from well A. The

equation is also dependent on reservoir pressure, fluid, and surrounding rock properties between two wells in a single-phase setting.

$$p_{wf(A)} = p_i - \sum_{j=1}^{i-1} \frac{141.2(q^{(j)} - q^{(j-1)})\mu B}{2kh} Ei\big[-\frac{\phi\mu c_t r^2}{0.001056k(t^{(i)} - t^{(j)})}\big] \tag{2.5}$$

The Ei-solution is often approximated by a logarithmic function, which is more widely used. However, it is important to note that the approximation is only valid for $\frac{t_D}{r_D^2} > 10$. At early time (low t_D), the infinite-acting radial flow response is not proportional to $log(t)$. As a result, features proposed by Liu (2013) which contain logarithmic 'flavor' might not be sufficient to match early time behavior.

In the fourth feature formulation shown in Eq. 2.6, exponential integral terms were introduced. The nonrate parameters outside the Ei-function are handled by the regression algorithm during training. The tricky part of using this term is the picking of the right parameter value inside the Ei-function which represents porosity (ϕ), viscosity (μ), total compressibility (c_t), and permeability (k). In our formulation, we used eight different parameter values ranging from 10^{-8} to 10^{-1}.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-8}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-7}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-6}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-5}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-4}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-3}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-2}}{(t^{(i)}-t^{(j)})}\big] \\ \sum_{j=1}^{i-1}(q^{(j)} - q^{(j-1)})Ei\big[-\frac{10^{-1}}{(t^{(i)}-t^{(j)})}\big] \end{bmatrix} \tag{2.6}$$

Suppose that x is proportional to $\frac{1}{(t^{(i)}-t^{(j)})}$. $Ei(-\frac{1}{x})$ can be approximated by $log(x)$ at high x value. At low x value, $Ei(-\frac{1}{x})$ has some exponential 'flavor'. An alternative form that is close to exponential integral function is a log over exponent term. This motivated us to try this alternative as the fifth feature formulation as shown in Eq. 2.7. In this formulation, we have four features from Eq. 2.3 and three additional exponential features for each phase in each well. In total, we have 14 features per well.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_o^{(j)} - q_o^{(j-1)}\right)/\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \frac{\left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{1}{\left(t^{(i)} - t^{(j)}\right)}}}) \\ \sum_{j=1}^{i-1} \frac{\left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{2}{\left(t^{(i)} - t^{(j)}\right)}}}) \\ \sum_{j=1}^{i-1} \frac{\left(q_o^{(j)} - q_o^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{3}{\left(t^{(i)} - t^{(j)}\right)}}}) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(q_w^{(j)} - q_w^{(j-1)}\right)/\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \frac{\left(q_w^{(j)} - q_w^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{1}{\left(t^{(i)} - t^{(j)}\right)}}}) \\ \sum_{j=1}^{i-1} \frac{\left(q_w^{(j)} - q_w^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{2}{\left(t^{(i)} - t^{(j)}\right)}}}) \\ \sum_{j=1}^{i-1} \frac{\left(q_w^{(j)} - q_w^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right)}{e^{\frac{3}{\left(t^{(i)} - t^{(j)}\right)}}}) \end{bmatrix} \tag{2.7}$$

2.2.3 Standardization

Standardization is an important part of building machine learning models that is often overlooked. In practice, it is recommended to standardize the data before feeding it into the model. Input standardization is a common requirement for many machine learning classifiers (Grus 2015). One of the possible consequences of not applying standardization is that the optimization (when performing gradient descent) can take a long time as the parameter value is oscillating before reaching minimum loss value. In the case where maximum optimization iteration is applied, the parameter might not be updated properly. Standardization shrinks the the shape of cost function contour into a circle in two dimensions (or sphere in three dimensions), allowing the optimized parameter to move directly to the center of the circle or the minimum. This helps the training process to converge faster.

$$X' = \frac{X - \overline{\mu}}{\sigma} \tag{2.8}$$

Standardization is performed by subtracting the mean of the feature from each data point and dividing by the standard deviation of the feature in the training set. The mean subtraction shifts the mean to zero and dividing by standard deviation converts the variance into a unit scale. The formulation is shown in Eq. 2.8. The mean and standard deviation from the training set are used to transform the development and test set. In Fig. 2.1, this process is labeled as "Transformation".

2.3 Machine/Deep Learning Algorithms

This section covers a brief description of ten machine/deep learning algorithms investigated in this study.

2.3.1 Ridge Regression

Ridge regression is a linear regression regularized with L2 norm. The matrix formulation is shown in Eq. 2.9 (Hastie et al. 2001), which is obtained from minimizing the sum of squares of residuals (Y and $\hat{Y}$).

$$W = (X^T X + \lambda I)^{-1} X^T Y \tag{2.9}$$

X is the input matrix of size m x n, where m is the number of data points and n is the total number of features. Y is the output vector size m. W is the vector of parameters (including bias term). λ is a hyperparameter that controls regularization strength. We need to adjust regularization strength such that the model is able to accurately model the training set (or minimize the bias) as well as perform well in other datasets with unseen inputs (or minimize the variance). This is a bias-variance trade-off problem. Trying to fit every single data point in the training set can lead to high variance or overfitting. The high variance can be reduced by penalizing W with λ as W grows larger (see Eq. 2.10). In other words, λ shrinks the contribution of each feature in X. Controlling the regularization strength is important to make sure that the model generalizes well. Eq. 2.10 shows the objective function of Ridge Regression, which consists of residual sum minimization and weight term penalization.

$$L(W, \lambda) = \underset{W}{\text{minimize}} \, ||XW - y||_2^2 + \lambda ||W||_2^2 \tag{2.10}$$

In the training period, given a pair of predictor X, ground truth Y, and user-defined λ, we can obtain W. In this research, optimum λ was obtained through hyperparameter grid search with $\lambda = [0.01, 1, 1, 2, 5, 10, 100, 1000]$. After obtaining W and λ, the prediction can be calculated using Eq. 2.11.

$$Y = XW \tag{2.11}$$

2.3.2 Kernel Ridge Regression

Kernel Ridge Regression (KRR) is a Ridge Regression with the application of kernelization. The kernel expands the original spaces to higher dimensions, allowing the model to learn a map function in the newly created spaces (Shawe-Taylor et al. 2004). Continuing from what we have seen in Section 2.3.1, we can write Eq. 2.9 as $W = X^T(XX^T + \lambda I)^{-1}y$ and for each predictor x' in the training set:

$$\hat{y} = W^T x' \tag{2.12}$$

$$\hat{y} = \left(X^T(XX^T + \lambda I)^{-1}y\right)^T x' \tag{2.13}$$

$$\hat{y} = y^T(XX^T + \lambda I)^{-1}Xx' \tag{2.14}$$

Now we can define a kernel $K_{ij} = \phi(x_i)^T\phi(x_j)$ that replaces XX^T and $\kappa(x) = K(x_i, x')$ that replaces Xx'). Kernel matrix K is defined as the inner product of expanded features $\phi(x_i)$ and $\phi(x_j)$ and can be linear or nonlinear. We can then update Eq. 2.14 to:

$$\hat{y} = y^T(K + \lambda I)^{-1}\kappa \tag{2.15}$$

Replacing x_i and x_j (elements in X) with $\phi(x)$ allows us to access the higher dimensions. We can compute K through inner products of $\phi(x_i)$ and $\phi(x_j)$ without calculating $\phi(x)$ explicitly. As an example, suppose we define the input features as:

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \end{bmatrix} \tag{2.16}$$

$$x_j = \begin{bmatrix} x_{j1} \\ x_{j2} \end{bmatrix} \tag{2.17}$$

Both features can be mapped to Eq. 2.18 below:

$$\phi(x) = \begin{bmatrix} x_{i1}x_{j1} \\ x_{i1}x_{j2} \\ x_{i2}x_{j1} \\ x_{i2}x_{j2} \end{bmatrix} \tag{2.18}$$

This research examined only linear as well as second-and third-order polynomial kernels. The loss calculation is given by Eq. 2.19. λ was optimized during hyperparameter search with the values included $= [0.01, 1, 1, 2, 5, 10, 100, 1000]$

$$L = \underset{W}{\text{minimize}}\frac{1}{N}\Big[\sum_{n=1}^{N}\big(y_n - \sum_{m=1}^{N} W_m K(x_n, x_m)\big)^2 + \lambda\sum_{n=1}^{N}\sum_{m=1}^{N} W_n W_m K(x_n, x_m)\Big] \tag{2.19}$$

2.3.3 Support Vector Regression

Support Vector Regression (SVR) aims to train parameter W and b in a linear equation (Eq. 2.20).

$$f(x) = WX + b \tag{2.20}$$

where X denotes the input matrix and $f(x)$ is the mapping function. This can be achieved by solving an optimization problem formulated in Eq. 2.21

$$\begin{aligned} &\underset{w,b,\xi^+ + \xi^-}{\text{minimize}} \frac{1}{2} W^T W + C \sum_{i=1}^{N} (\xi_i^+ + \xi_i^-) \\ &\text{subject to: } -\epsilon - \xi_i^- \leq y_i - W^T x_i - b \leq \epsilon + \xi_i^+ \\ &\qquad \xi_i^+ \geq 0 \text{ and } \xi_i^+ \geq 0 \end{aligned} \tag{2.21}$$

where ξ^+ and ξ^+ are the penalty terms in the upper and lower soft margin ϵ for data points that cannot satisfy the optimization constraints. These are often called slack variables. Hyperparameter C controls the tolerance penalty (as a result of soft margin violations, i.e. data points that are further than ϵ from the main vector). Values of C included in this research during hyperparameter search are $[1, 10, 100, 1000, 10000]$, all using linear kernel. Eq. 2.21 is often referred to the primal formulation. The problem is often easier to solve in its Lagrangian dual formulation (Smola 2003), see Eq. 2.22.

$$\begin{aligned} Ł := &\frac{1}{2} W^T W + C \sum_{i=1}^{N} (\xi_i^+ + \xi_i^-) - \sum_{i=1}^{N} (\eta \xi_i^+ + \eta' \xi_i^-) \\ &- \sum_{i=1}^{N} \alpha_i (\epsilon + \xi_i^- - y_i + W^T x_i + b) \\ &- \sum_{i=1}^{N} \alpha_i' (\epsilon + \xi_i^+ + y_i - W^T x_i - b) \end{aligned} \tag{2.22}$$

where L is the Lagrangian and η, η', α, and $\alpha' \geq 0$ are Lagrange multipliers. In the optimum condition, the derivatives of L with respect to primal variables are equal to zero, see Eq. 2.23, 2.24, and 2.25.

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N} (\alpha' - \alpha) = 0 \tag{2.23}$$

$$\frac{\partial L}{\partial W} = W + \sum_{i=1}^{N} (\alpha' - \alpha) x_i = 0 \tag{2.24}$$

$$\frac{\partial L}{\partial \xi_i^+} = \sum_{i=1}^{N} C - \alpha_i' - \eta' \tag{2.25}$$

We can substitute Eq. 2.23, 2.24, and 2.25 to Eq. 2.22 and obtain a dual optimization form. We can then solve for α and α' at the optimum condition. When an optimum is achieved, we can calculate W in Eq. 2.24 by substituting α and α'. Finally, the main equation (Eq. 2.20) can be expressed as follows:

$$f(x) = WX + b = \sum_{i=1}^{N}(\alpha' - \alpha)x_i + b \tag{2.26}$$

2.3.4 Stochastic Gradient Descent Regression

When performing gradient descent iteration, one can define the size of dataset in each parameter update step. In batch gradient descent, all data points are involved in each parameter update step. If the number of data points is large, using batch gradient descent can take a long time to converge. On the other hand, Stochastic Gradient Descent Regression (SGD Regression) performs parameter update each time the model encounters a training example.

$$\theta_j := \theta_j + \alpha\big(y^{(i)} - \hat{y}^{(i)}(x)\big)x_j^{(i)}, \text{ for every i data point and j feature} \tag{2.27}$$

where θ is the parameter, $y^{(i)}$ is ground truth/label, $\hat{y}^{(i)}(x)$ is the prediction outcome, α is the learning rate, and $x_j^{(i)}$ is the feature input j (a component in matrix X). The structure of the main equation (X to y mapping) is the same as Ridge Regression (see Eq. 2.9). Similar to Ridge and Kernel Ridge Regression, regularization strength control $\lambda = [0.01, 1, 1, 2, 5, 10, 100, 1000]$ is optimized during hyperparameter search.

2.3.5 Decision Tree

Decision Tree (also called Regression Tree in regression problems) is a supervised learning model that creates branches of decision nodes. Decision Tree learns a dataset by making the best split of features in the training set into subsets based on the homogeneity of each new subset. The process goes on until a stopping criterion (e.g. tree size or number of observation per node) is reached (Breiman et al. 1984). We can control how the decision tree algorithm splits nodes.

Suppose we have input $X = [x_1, x_2]^T$ and target y in the training set. We want to divide X into J distinct and nonoverlapping regions $R_1, R_2, ..., R_J$. Every x_1 or x_2 value that falls into region R_j has one prediction y_{Rj}, which is the average response values in the region R_j.

The measure of the quality of a split used in this study is mean squared error, which minimizes the L2 loss in each node, as shown in Eq. 2.28.

$$L = \sum_{n=1}^{N}\sum_{i:x_i \in R}(\hat{y}_{Rj} - y^{(i)})^2, \tag{2.28}$$

where $y_{Rj} = \frac{1}{N_J}\sum_{i \in J} y^{(i)}$, or the mean response for the training observation within the jth box (James et al. 2013). We performed binary splitting by selecting predictor x_j (in this case either x_1

or x_2) and cutpoint s such that x_j is split into two regions: $\{X|x_j < s\}$ and $\{X|x_j > s\}$. We scan through all possible values of s in each x_j and select s and j that minimize Eq. 2.29.

$$L = \sum_{n=1}^{N} \sum_{i:x_i \in R_1(j,s)} (\hat{y}_{R1} - y^{(i)})^2 + \sum_{n=1}^{N} \sum_{i:x_i \in R_2(j,s)} (\hat{y}_{R2} - y^{(i)})^2 \tag{2.29}$$

where $\hat{y}_{R1}$ is the mean response for $y^{(i)}$ in R_1 and $\hat{y}_{R2}$ is the mean response for $y^{(i)}$ in R_2. We can think of R_1 as the left region and R_2 as the right region in a one-dimensional axes. The process is continued until a stopping criterion (e.g. tree size or number of observation per node) is reached. At this point, the training process stops and a tree-based model has just been built.

In this study, we chose the best classifier among following maximum tree depth = $[5, 10, 30, 50, None]$. "None" means that the tree can grow as deep as possible until other stopping criteria are met.

2.3.6 Random Forest

Decision Tree grows a single tree to map input X and output y. Random Forest, on the other hand, is an ensemble algorithm that builds multiple classifiers during training. This study used Decision Tree as the classifier. In each tree, the model picks random number of features and corresponding label. Then, each tree performs splitting until a criteria is met (either mean square error or maximum tree depth), the same way as Decision Tree algorithm in Subsection 2.3.5. In the testing time, given an input, Random Forest outputs multiple values from all the decision trees and then averages them into a single final output (or majority vote in classification problem). The averaging process can be useful as a regularizer to improve the test accuracy. The generic formula is shown in Eq. 2.30

$$\hat{f}(X) = \frac{1}{B} \sum_{b=1}^{B} f_b(X) \tag{2.30}$$

where $\hat{f}(X)$ is the output of random forest mapping function, $f_b(X)$ is the output of individual tree function, and B is the number of classifiers or estimators (Hastie et al. 2001). Two hyper-parameters were used in the study: number of classifiers ($B = [2, 3, 5, 10, 20]$) and individual tree maximum depth ($[5, 10, 30, 50, None]$).

2.3.7 AdaBoost

AdaBoost creates T weak learners h_t (in this study trees) and boosts these weak learners to stronger ones. It does so by assigning more weight to data points whose error is high. In this study, we used linear loss function to update the weights. In the next iteration, the algorithm will try to avoid mispredicting these data points as it would greatly increase the penalty/error. Similar to Random Forest, the final prediction H is computed as a weighted average of the weak learners' prediction. Eq. 2.31 shows the general form of AdaBoost.

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.31}$$

where $H(y)$ is the output of Adaboost mapping function, α_t is weight assigned to each classifier, $h_t(x)$ is output of the weak classifier/estimator/learner (Hastie et al. 2001).

First, we pick K random samples with replacement and construct a Decision Tree regressor h_t as in Section 2.3.5. We pass every $x^{(i)}$ to obtain $\hat{y}^{(i)}$. The loss can be computed using linear loss function given in Eq. 2.32.

$$L = \frac{|\hat{y}^{(i)} - y^{(i)}|}{D} \tag{2.32}$$

where $D = sup|\hat{y}^{(i)} - y^{(i)}|$. Next, we compute the average loss $L = \sum_{k=1}^{K} L_k p_k$ where $p_k = \frac{w_k}{\sum_{i=1}^{N} w_i}$, which is the probability that training sample i is in the training set (Drucker 1997). We then define β as a measure of confidence, which is formulated as follows.

$$\beta = \frac{L}{(1 - L)} \tag{2.33}$$

The next w_i is updated using Eq. 2.34.

$$w_i := w_i \beta (1 - L) \tag{2.34}$$

Finally, for every predictor t, the prediction is computed using Eq. 2.35.

$$H(x) = \frac{\sum_t log(\frac{1}{\beta_t} h_t(x)}{\sum_t log(\frac{1}{\beta_t}} \tag{2.35}$$

We optimized the number of estimators through grid search; the values are given as follows $= [2, 5, 10, 20, 50, 100]$.

2.3.8 Gradient Boosting

Similar to AdaBoost, Gradient Boosting (or Gradient Boosting Machine) uses an ensemble of weak prediction classifiers (decision trees, in this study). Each individual tree is trained as in Section 2.3.5. The model trains the tree one at a time and calculates mean squared error loss. The next step is gradient descent. But the unique step here is instead of updating parameter value, the gradient descent updates a tree function by modifying its parameters so that it moves to the right gradient (Friedman 2011).

Given training data $x^{(i)}$ and $y^{(i)}$, we want to build a model $F(x)$ that minimizes square loss. For each iteration, we can improve the current model by adding a new tree model $h(x)$, such that:

$$F(x^{(i)}) + h(x^{(i)}) = y^{(i)} \tag{2.36}$$

Or, equivalently:

$$h(x^{(i)}) = y^{(i)} - F(x^{(i)}) \tag{2.37}$$

Next, we can form a loss function $L(y^{(i)}, F(x^{(i)}))$, which is a squared error of the residual form in Eq. 2.37:

$$L(y^{(i)}, F(x^{(i)}) = \frac{(y^{(i)} - F(x^{(i)})^2}{2} \tag{2.38}$$

In order to minimize the loss, we implement the gradient descent. The formulation is given in Eq. 2.39.

$$F(x^{(i)}) := F(x^{(i)}) - \alpha \frac{\partial J}{\partial F(x^{(i)})}, \text{ where } \frac{\partial J}{\partial F(x^{(i)})} = F(x^{(i)}) - y^{(i)} \tag{2.39}$$

where α is the learning rate (set to 0.1 in this study). Notice that the negative gradient $-\frac{\partial J}{\partial F(x^{(i)})}$ is equal to tree classifier h in Eq. 2.37. We can fit a tree h such that its gradient is equal to $-\frac{\partial J}{\partial F(x^{(i)})} = y^{(i)} - F(x^{(i)})$.

Grid search is performed over the number of estimators in the study: $[2, 5, 10, 20, 50, 100]$.

2.3.9 k-Nearest Neighbors Regression

k-Nearest Neighbors uses local neighborhood points to make a prediction. In more detail, distance (Euclidean distance in this study) is calculated from a query point $x^{(i)}$ in the test set to each data point $x'^{(i)}$ in the training time, see Eq. 2.40.

$$D(x, x') = \sqrt{\sum_{i=1}^{N} (x^{(i)} - x'^{(i)})^2} \tag{2.40}$$

Next, we sort the distance and pick k closest data points. We call the k points closest points in the training set that are closest to x the set G. The prediction during testing time is calculated by local interpolation of the nearest neighbors in G, see Eq. 2.41.

$$\hat{y}^{(i)} = \frac{1}{K} \sum_{j \in G}^{N} y_j \tag{2.41}$$

The number of nearest neighbors (k) is a parameter that users can adjust; this study used: $[2, 3, 5, 10, 20]$ neighbors during optimization. k-Nearest Neighbors is often called instance-based learning algorithm as it has to memorize the training instances instead of explicitly learn a model. As a result, for large data sets application, k-Nearest Neighbors is computationally expensive.

2.3.10 Neural Network with Fully Connected Layers

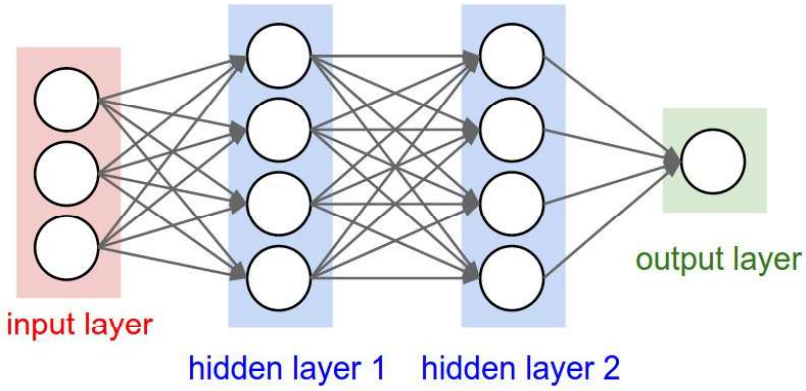

Figure 2.3: Visualization of fully connected layers structure (taken from [11]).

Neural Network is a network of interconnected elements, which takes on input, activates, and then passes it to the next layer (see Fig. 2.3). There are broad choices of connection types, but this investigation experimented with fully connected layers, where every neuron from the input is mapped to every output neuron. Also in this study, the model used 'ReLU' as an activation function (see Eq. 2.43) in every layer. For each layer l, the mapping functions are given as follows:

$$z^{[l]} = W^{[l]T} a^{[l-1]]} + b^{[l]} \tag{2.42}$$

$$a^{[l]} = max(0, z^{[l]}) \tag{2.43}$$

where $W^{[l]}$ and $b^{[l]}$ are the weight matrix and bias in layer l, respectively. $a^{[l]}$ is the output in layer l. The weights W were initialized such that it has zero mean an a specific variance (Glorot and Bengio 2009):

$$Var(W) = \frac{2}{(n_{in} + n_{out})} \tag{2.44}$$

where n_{in} and n_{out} are the number of inputs and outputs of layer l, respectively. All the bias terms b were initialized to zero.

We passed the input x as $a^{[0]]}$ in the first layer and propagated the input to the last layer to obtain $\hat{y}$. The loss was computed using squared-loss function. The model was optimized using Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), a quasi-Newton optimization algorithm as solver. The model is regularized by L2 norm with $\lambda = 0.0001$. We optimized the architecture of the model by tuning the number of hidden layers and neurons in each layer. The list of hyperparameters is as follows: [(10), (20), (50), (100), (1000), (20,5), (20,20), (20,5,10), (20,5,20), (50,10,20),

(100,20,50), (1000,50,100), (1000,200,1000), (1000,50,400,600)]. As an example, (1000,50,100) indicates that there are three layers; first layer has 1000 neurons, second layer has 50 neurons, and third layer has 100 neurons.

2.4 Test Case Definition

2.4.1 Case 1

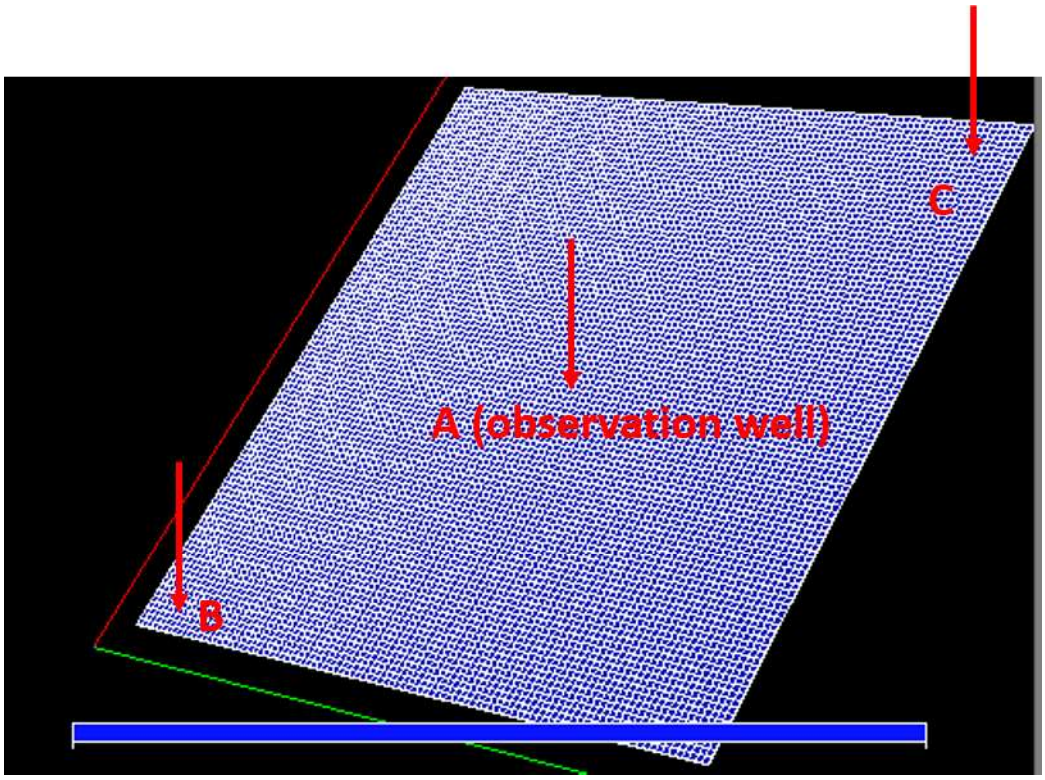

Figure 2.4: Well locations in Case 1.

In Case 1, three producers are far away from each other (see Fig. 2.4 for visualization and Table 2.4.1 for the exact coordinates).

Table 2.3: Well locations in Case 1

Block Coordinate	Value
Well A location	(50, 50)
Well B location	(3, 3)
Well C location	(88, 88)

Figure 2.5 shows the visualization of dataset in Case 1. This case assumes a dataset with no noise

and skin damage. All three wells are producing at low water cut (up to 20%). Average reservoir pressure change during training and testing is small due to the size of the reservoir.

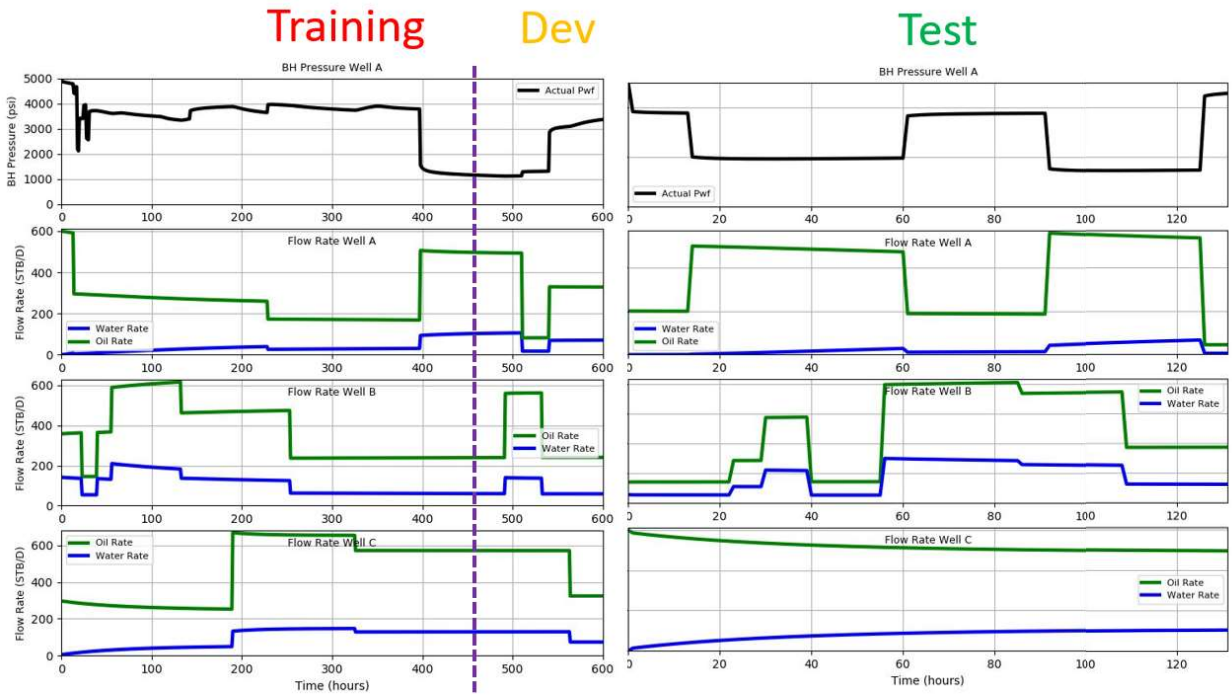

Figure 2.5: Visualization of training, development, and test sets in Case 1.

2.4.2 Case 2

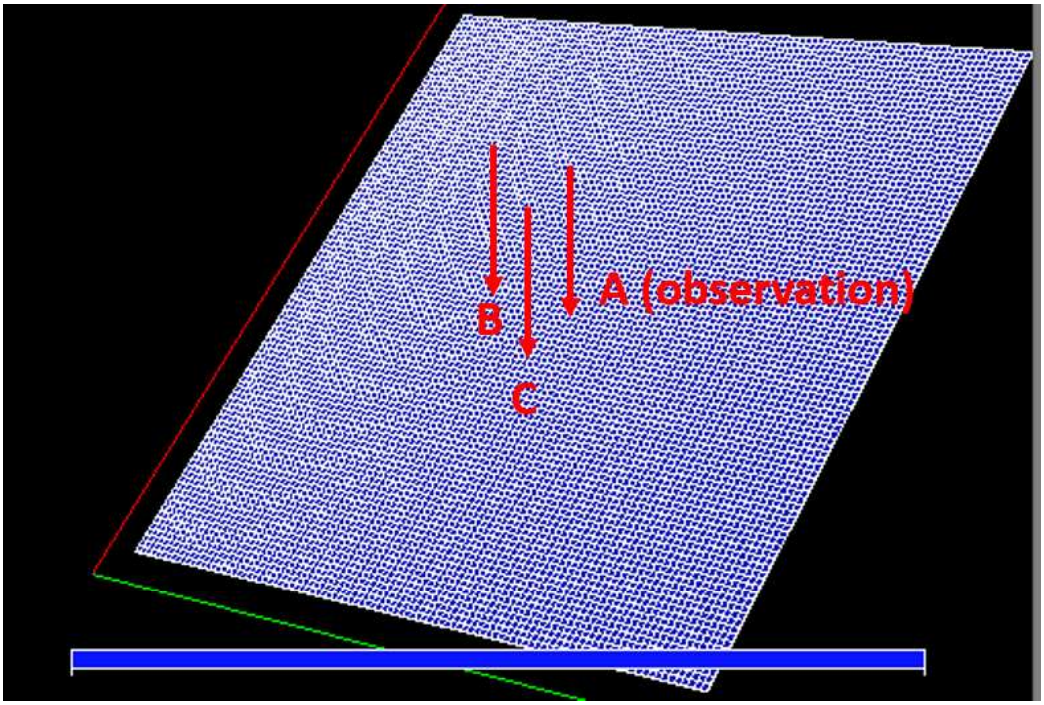

Figure 2.6: Well coordinate in Case 2.

Case 2 is similar in to Case 1. The main difference is that the wells are now closer to each other. Another difference is that three wells produce high water cut (up to 75%). See Fig. 2.6 for well and reservoir visualization and Fig. 2.7 to see bottom hole pressure and flow rates variations.

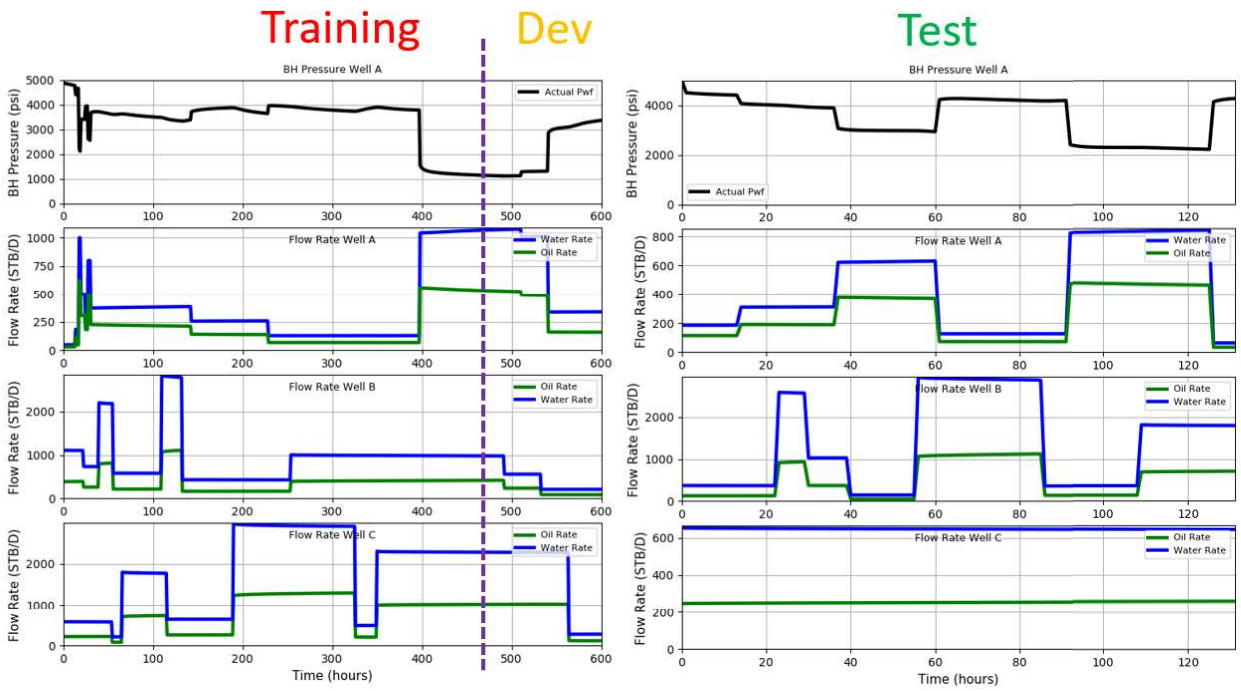

Figure 2.7: Visualization of training, development, and test sets in Case 2.

Table 2.4: Well coordinate in Case 2

Block Coordinate	Value
Well A location	(50, 50)
Well B location	(47, 50)
Well C location	(50, 47)

Chapter 3

Two-Phase Bottom Hole Pressure Prediction

In this chapter, we applied the workflow and methods discussed in Chapter 2 to predict bottom hole pressure on well A, given oil and water rates from three production wells. The results were presented on case by case basis using five feature formulations and ten different algorithms. All five formulations were summarized as follows:

- Feature Formulation 1 used raw oil and water rates as in Eq. 2.1.
- Feature Formulation 2 (introduced by Liu(2013) for a single-phase flow) is shown in Eq. 2.3.
- Feature Formulation 3 (expansion of Feature Formulation 2 to capture water phase) is shown in Eq. 2.4.
- Feature Formulation 4 (containing Ei-function terms) is shown in Eq. 2.6.
- Feature Formulation 5 (containing exponential terms) is shown in Eq. 2.7.

3.1 Case 1

3.1.1 Feature Formulation Performance Comparison

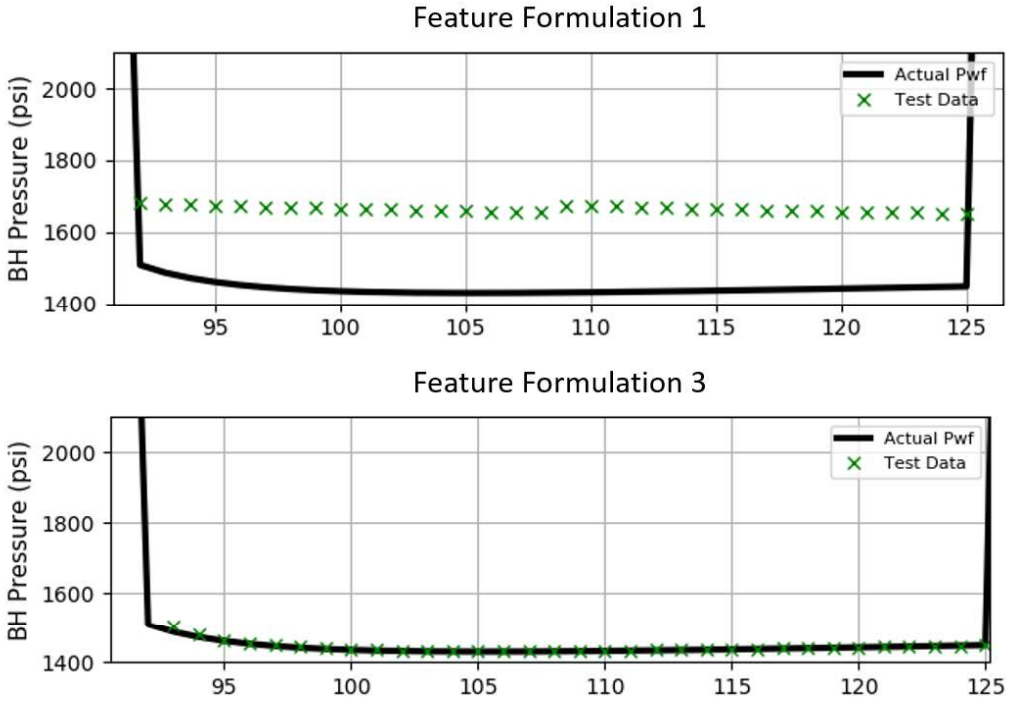

Figure 3.1: Prediction result comparison between Feature Formulation 1 and 3, both using Support Vector Regression (Case 1).

Using raw data as the feature is the easiest and fastest way of building a machine learning model. However, the resulting bottom hole prediction was poor and far from the actual data. Unlike feature-based model, machine learning model with raw data input has no knowledge about the physics of the flow. A kernel function can map a raw input to higher dimensions, but still cannot capture the convolution nature of pressure. The top plot in Fig. 3.1 shows that a model with raw input data tended to have flat bottom hole prediction and did not capture the transient effect well. In contrast, Feature Formulation 3 better matched the actual bottom hole pressure, as shown in the lower figure. Feature Formulation 3 contains total eight features per well that represent both oil and water phases.

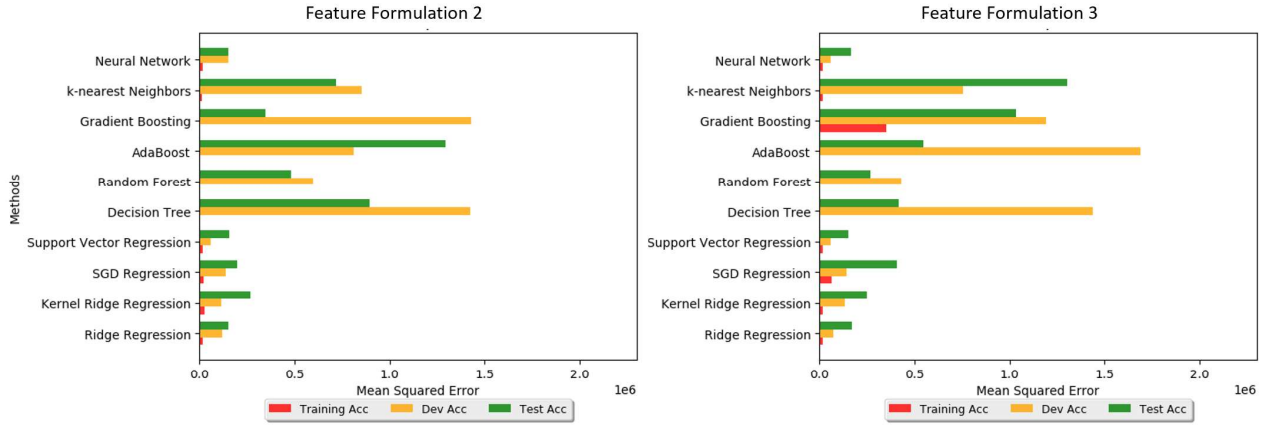

Figure 3.2: Mean squared errors comparison between Feature Formulation 2 and Feature Formulation 3 (Case 1).

The left figure in Fig. 3.2 shows training, development, and test mean squared errors for Feature Formulation 2, which contains four features representing only oil phase. If we only look at Neural Network, Ridge Regression, Kernel Ridge Regression, and Support Vector Regression, adding four additional features for water phase into the input matrix slightly made the development and test scores better, as shown in the right figure. This is intuitive as the bottom hole pressure is affected by both oil and water phases. The reason for the small difference in performance is partly because of low water cut; the water phase only accounted for up to 20% of total liquid rate. Another reason is that the water and oil properties in this case are not far different. It turned out that both feature formulations did not work well for k-Nearest Neighbors, Gradient Boosting, AdaBoost, and SGD Regression due to overfitting.

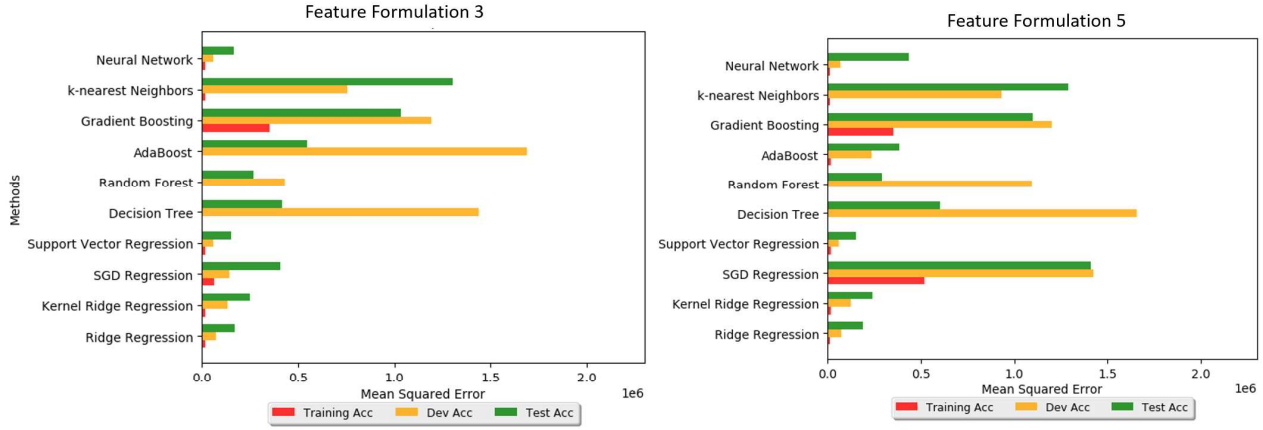

Figure 3.3: Mean squared errors comparison between Feature Formulation 3 and Feature Formulation 5 (Case 1).

Fig. 3.3 shows the comparison between the results of Feature Formulations 3 and 5. We observed similar results using Support Vector Regression, Ridge Regression, and Kernel Ridge Regression. Feature Formulation 5 has three additional exponential features which aimed to capture the nonlogarithmic early time response, which is a function of well distance and time. The fact that the three producers were far away from each other in Case 1 was the reason behind the similarity in results. Later in the Case 2 discussion, we will see how these two formulations produced different results.

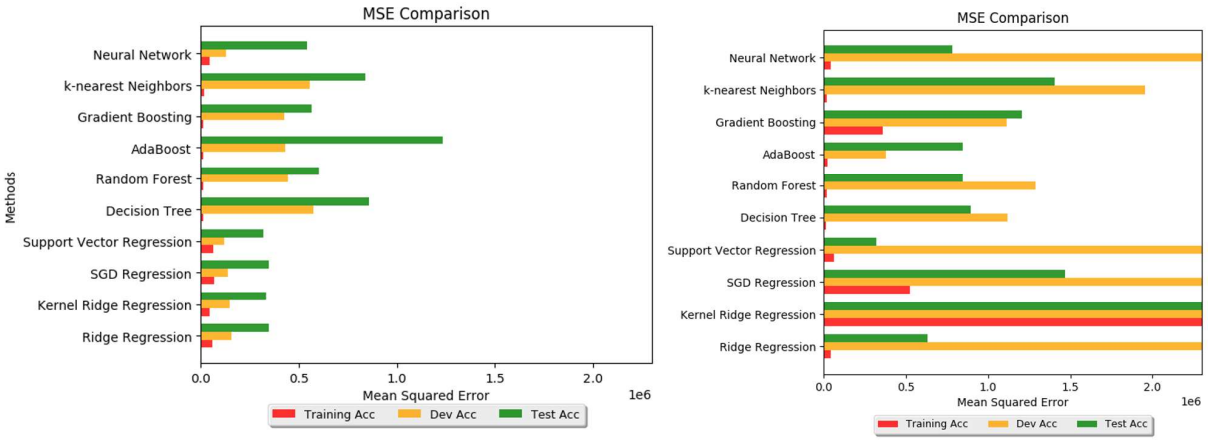

Figure 3.4: Mean squared errors comparison with Feature Formulation 4 with low parameter values ranging from 10^{-8} to 10^{-1} (left) and high parameter values from 10^{-3} to 10^{4} (right).

Each element in Feature Formulation 4 contains an exponential integral term, which is the analytical solution to the diffusivity equation. However, determining the correct parameter values in Eq. 2.6 is very tricky. These values can be considered as hyperparameters that need to be adjusted by experimenting in the training and dev set. Fig. 3.4 shows how two different parameter value ranges affect the result. The right figure shows that setting too high parameter values led to high mean squared error.

Adding more features with more parameter values increases the chance of getting a better result. However, there is a trade-off between the model accuracy and runtime. Building features from convolutions of flow rates can be time time-consuming, especially if we are working on a long dataset with high interval frequency. In this study, we assumed no prior understanding about the correct parameter value, which is a function of rock and fluid properties . It would be useful if porosity (ϕ), viscosity (μ), total compressibility (c_t), and permeability (k) are known prior the training so that a the correct parameter value can be approximated.

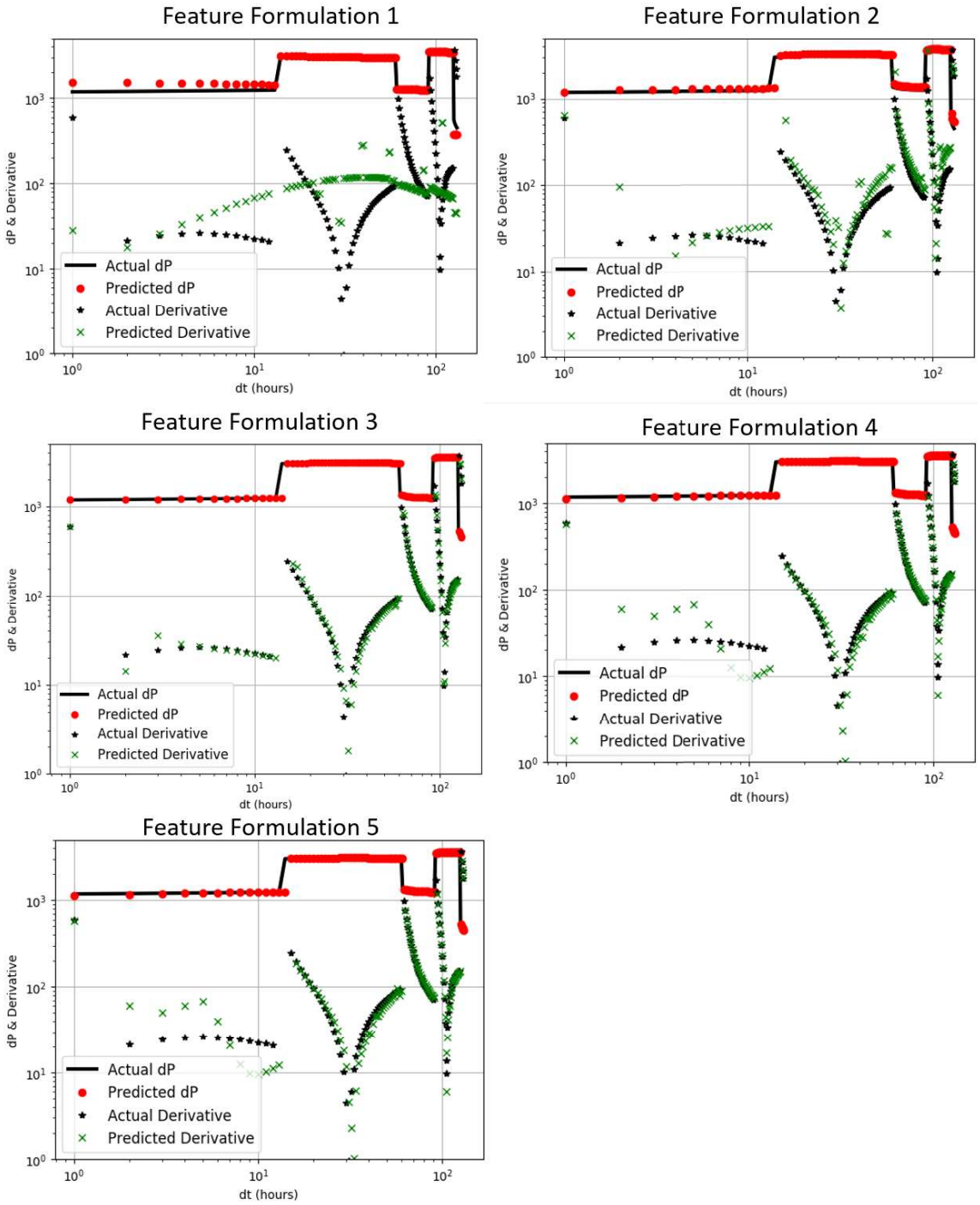

Figure 3.5: Pressure derivative results comparison of different feature formulations.

While mean squared error values can be useful in making a comparison, these are not the only

things to look at. We were also interested in the resulting derivative plots. It turned out that despite having a low mean squared error, Feature Formulation 1 had poor derivative result, as shown in Figure 3.5. The same figure also shows how adding features help improve the resulting derivative plots. Adding four features for oil phase in Feature Formulation 2 gave the correct trend. Adding four more additional features for water phases in Feature Formulation 3 resulted in an even better match. Feature formulation 4 and 5 have similar performance.

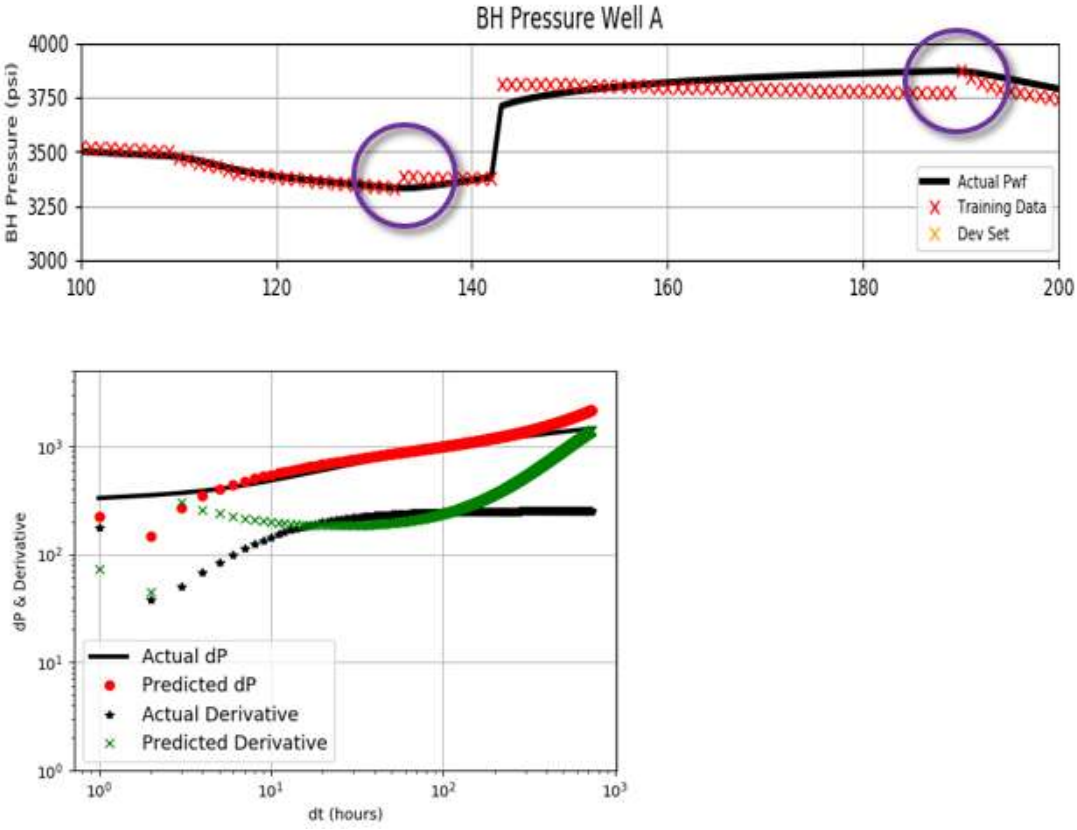

Figure 3.6: Top: Bottom hole pressure prediction using Fully-connected Neural Network with raw data input. Bottom: Derivative plot using the same algorithm and formulation.

In Fig. 3.2, 3.3, and 3.4, we can see that fully-connected Neural Network performed well given the right features. However, feeding the fully-connected Neural Network model with only raw data did not yield desired results and was worse in performance compared to the feature-based approaches (see Fig. 3.6). This Neural Network model could be useful in estimating a rough value of pressure but was not able to carry important information (such as infinite-acting behavior) in the derivative plot. Another study showed that Recurrent Neural Network (RNN) was able to solve the problem and retained important information in the derivative plot (Tian 2017).

3.1.2 Algorithm Performances Comparison

Fig. 3.7 shows that Support Vector Regression, Ridge Regression, and Neural Network consistently outperformed the other algorithms in every feature formulation. On the other hand, k-Nearest Neighbors, Decision Tree, Gradient Boosting, Random Forest, and AdaBoost all performed poorly. The big difference between training and development scores indicated severe overfitting problems. In Section 3.1.4, we will see the visualization outputs and a more detailed explanation about the problem. In general, this study found that tree-based and ensemble algorithms are not suitable for production this time-series problem.

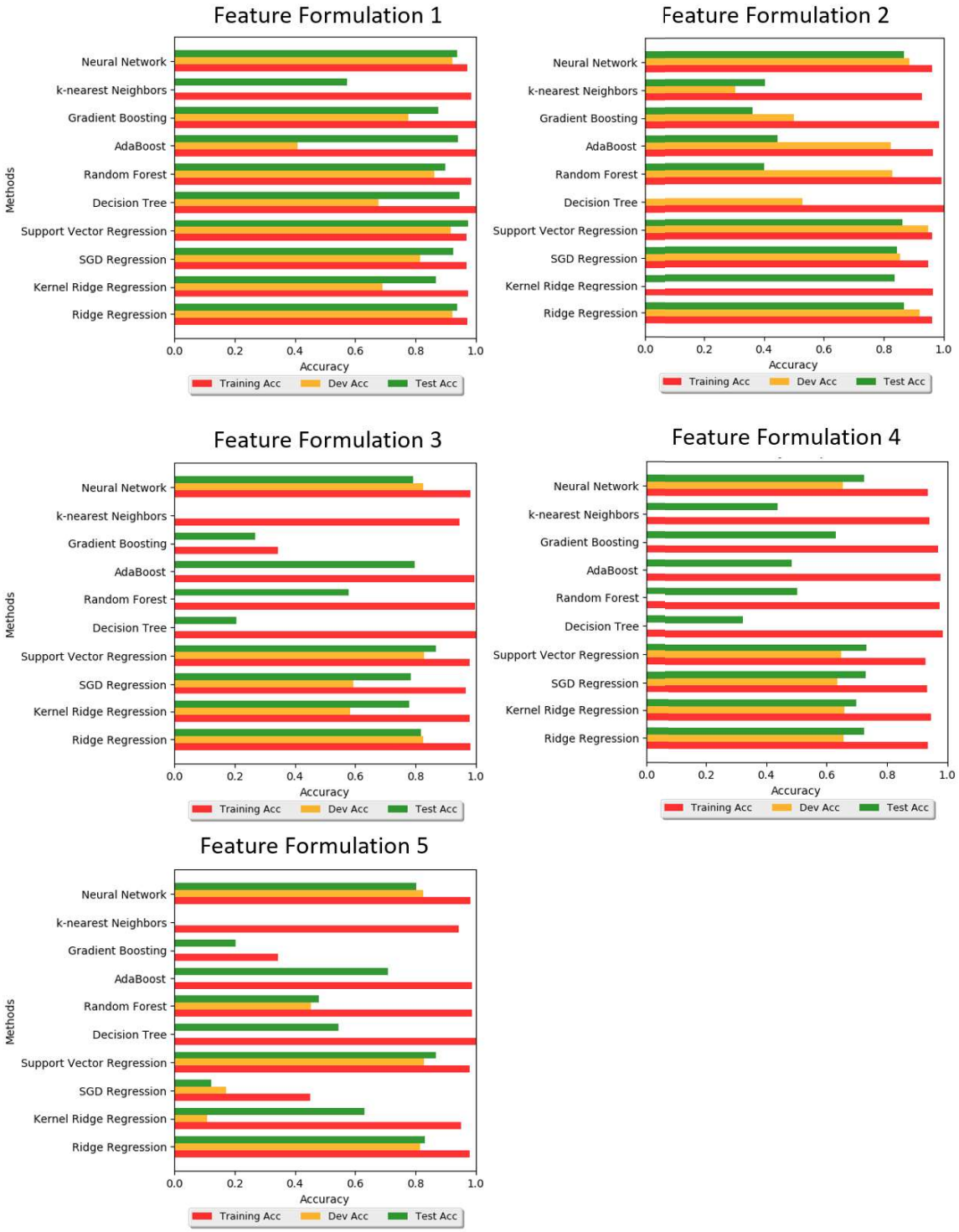

Figure 3.7: R-squared scores of ten different algorithms and five features formulations. Note that this figure is intended to compare different algorithms with the same feature formulation. It is not appropriate to compare scores between different feature formulations.

In the next section, we will see some visualizations using algorithms that worked well in the experiment.

3.1.3 Highlights of Good Performing Algorithms

Support Vector Regression (SVR) performed well with Feature Formulations 3, 4, and 5. Fig. 3.8 shows the visualization of the result with Feature Formulation 3. It is very clear that SVR generalized well and was able to achieve high scores in training, development, and test sets. SVR also captured the transient period after every flow rate change event quite well. Note that this section only shows the result using Feature Formulation 3 and not Feature Formulation 4 and 5 as they have roughly similar performances. We will see in Case 2 where the results of these feature formulations differ from each other.

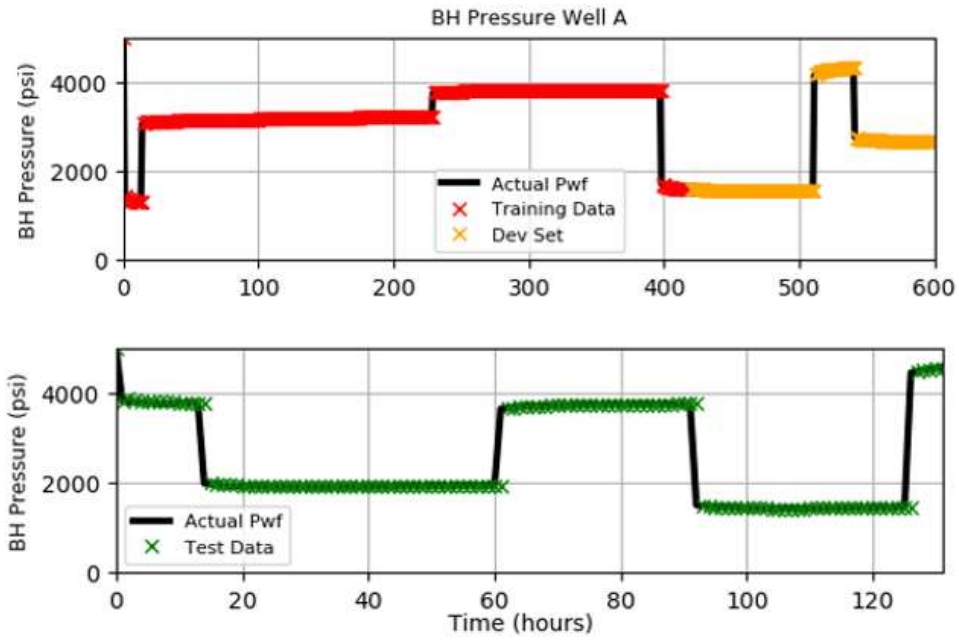

Figure 3.8: Predictions using Support Vector Regression with Feature Formulation 3 in training, development, and test sets (Case 1).

The good performance in the development set shows that the algorithm can be used to forecast bottom hole pressure right after the end of the training set. The algorithm also performed well in the test set, that is not a continuation of the training and development set. We can imagine the test set as a production profile after a long shut-in.

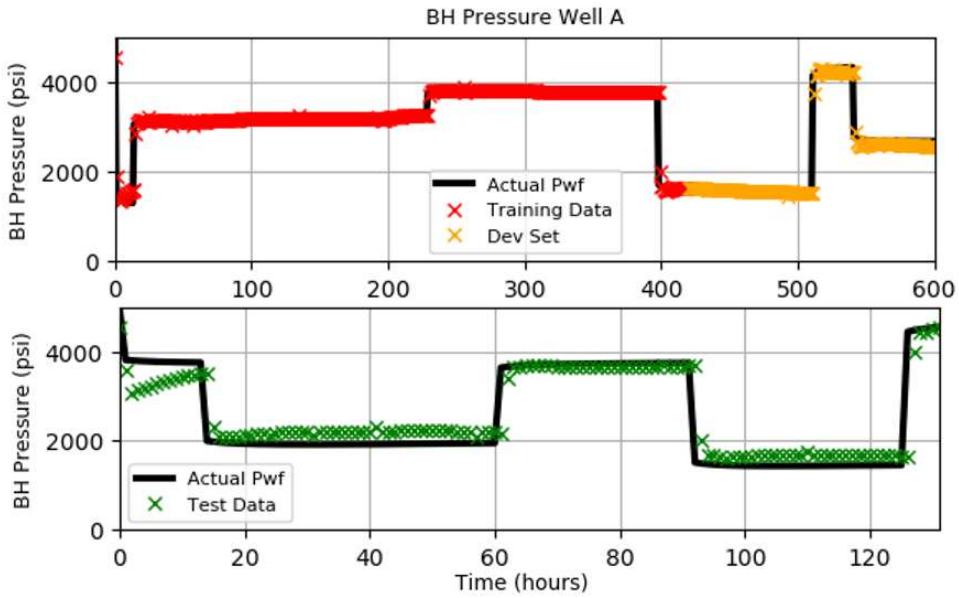

Figure 3.9: Predictions using Ridge Regression with Feature Formulation 3 in training, development, and test sets (Case 1).

Ridge regression also generally performed well, with the exception of early time in the test set. The difference between actual data and prediction was caused mainly by water cut change from zero to around 10 percent that was not well captured by the model. After 125 hours, the water cut was stable at 10%. In the training set, the model was trained mostly at water cut = 20% and this was the cause of the bias. This can be addressed by adding more data in the range that was underrepresented in the training set.

Neural Network did not perform as well as SVR and Ridge Regression, but it was able to get the correct trend at the least. Similar to Ridge Regression, the model could not learn water-cut change well. The result is shown in Fig. 3.10.

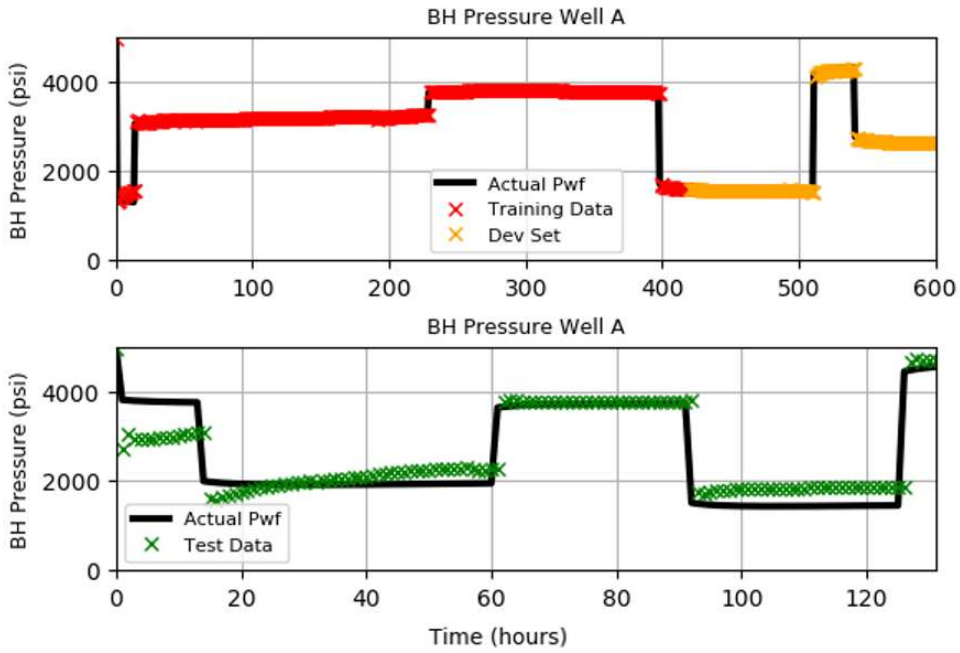

Figure 3.10: Predictions using fully-connected Neural Network with Feature Formulation 3 in training, development, and test sets (Case 1).

3.1.4 Highlights of Bad Performing Algorithms

In contrast to good performers in Section 3.1.3, this section highlights some algorithms that did not perform well. The first example is k-Nearest Neighbors prediction, shown in Fig. 3.11.

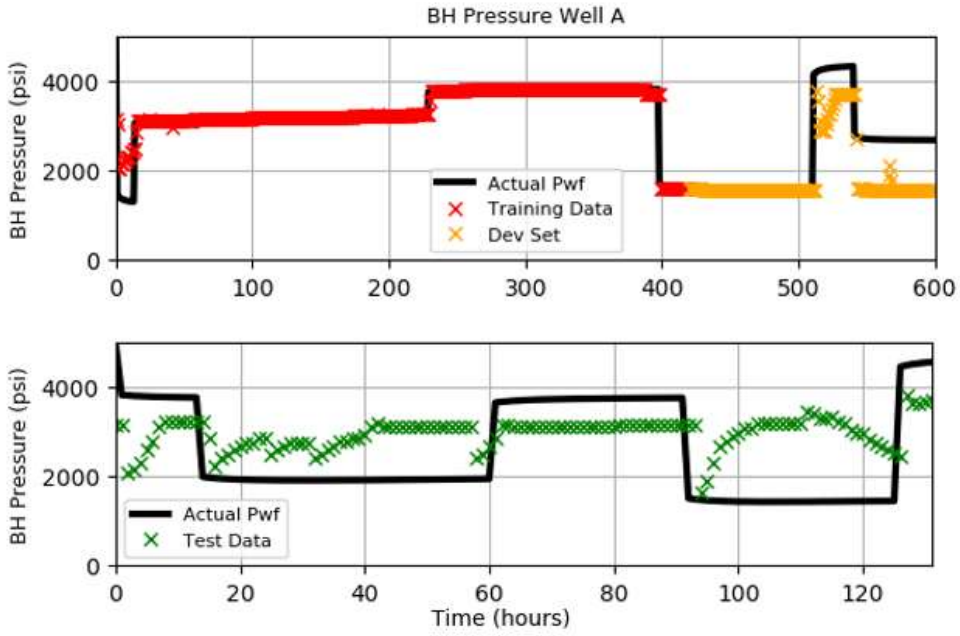

Figure 3.11: Predictions using k-Nearest Neighbors with Feature Formulation 3 in training, development, and test sets (Case 1).

The algorithm performed quite well in the training set (with the exception of the early time production). But, k-Nearest Neighbors did not generalize well and showed poor performance in both development and test sets. The algorithms overfitted the training set and did not learn the essential pattern of the dataset that we desired. Even with the help of the features, which gave the algorithm some 'clues' about the physics of the flow, the algorithm still could not generalize well. In addition, k-Nearest Neighbors is often called a 'lazy' algorithm as it requires mapping from each data point in test (or development) set to each data point in training set. This can be an issue when deploying this algorithm in a big data set.

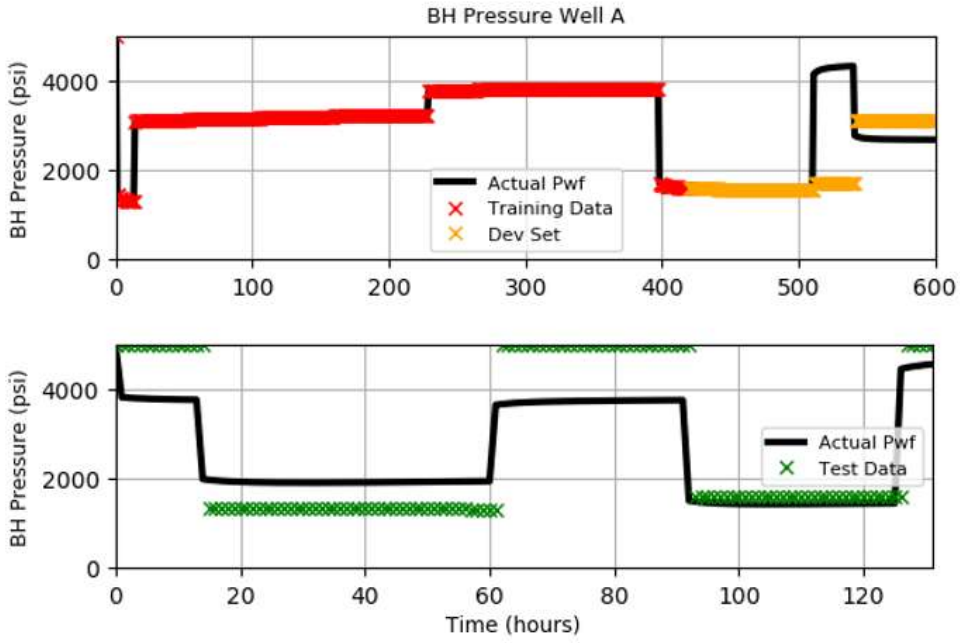

Figure 3.12: Predictions using Decision Tree with Feature Formulation 3 in training, development, and test sets (Case 1).

Another bad performing algorithm is Decision Tree, see Fig. 3.12. Similar to k-Nearest Neighbors, Decision Tree suffered from the same overfitting problem.

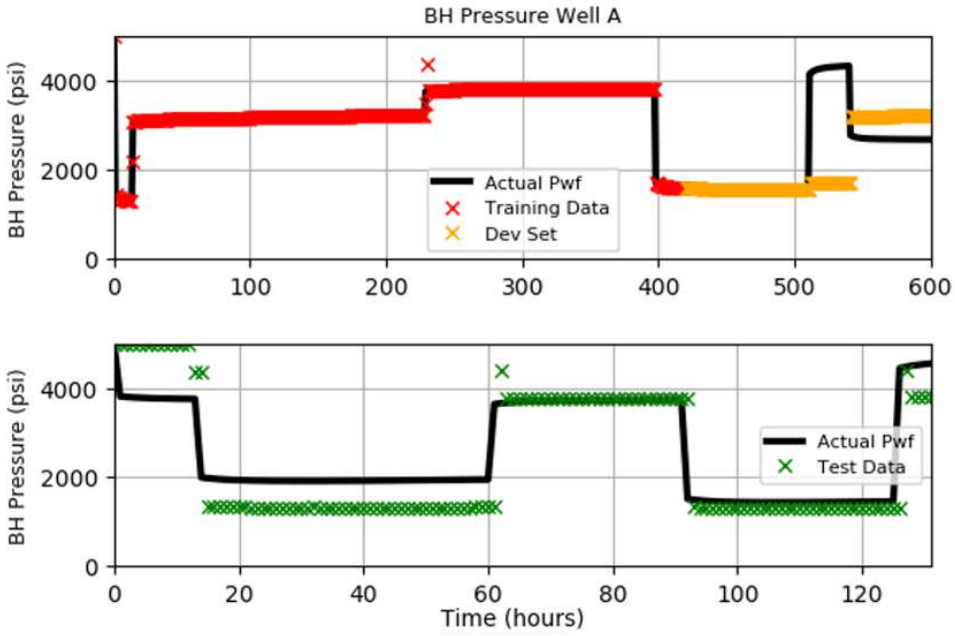

Figure 3.13: Predictions using Random Forest with Feature Formulation 3 in training, development, and test sets (Case 1).

Random Forest is commonly used to solve overfitting problem in the Decision Tree algorithm as it takes an average of many tree classifiers, which can reduce model variance. In Fig. 3.13, Random Forest slightly increased the test performance, but did not solve the problem entirely.

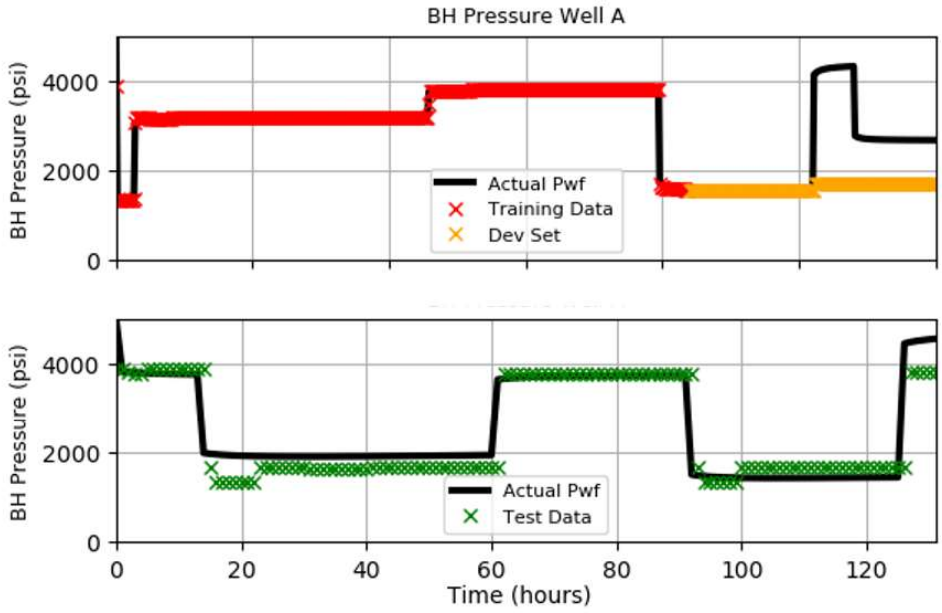

Figure 3.14: Predictions using AdaBoost with Feature Formulation 3 in training, development, and test sets (Case 1).

Similar to Random Forest, AdaBoost also contains an ensemble of trees. The main difference is that, when boosting each tree, AdaBoost tends to increase the variance of the model. This exacerbated the problem as the individual tree model had high variance or overfitting issue already. As expected, in Fig. 3.14, the performance of AdaBoost was poor.

3.2 Case 2

In this case, we were interested to see the performance of different algorithms and feature formulations if the well distance is close and the water cut is high.

3.2.1 Feature Formulations Performance

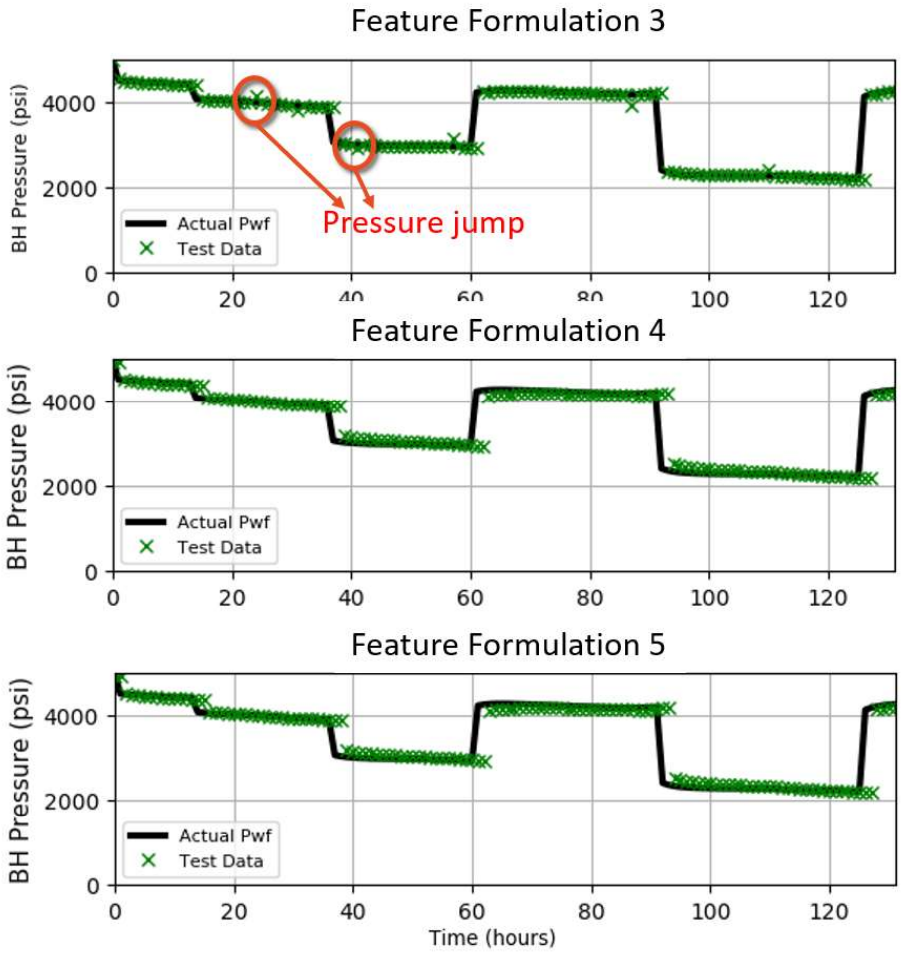

Figure 3.15: Resulting bottom hole pressure prediction using Feature Formulation 3, 4, and 5 in Case 2, all using Support Vector Regression.

Feature Formulation 3 performed well in general, except at the early time, see the top figure in Fig. 3.15. It was observed that the predicted pressure on well A jumped right after flow rate B or C changed. This was something we did not see in Case 1. If we look at Eq. 2.5, pressure response on observation well is dependent on well distance r. Also, *log* approximation to the Ei-function that is captured in Feature Formulation 3 is only valid in certain $\frac{t_D}{r_D^2}$.

This phenomenon can be clearly explained in Fig. 3.16. The term x in the Ei-function is proportional to $\frac{t}{r^2}$. If t is small, x will also be small and Ei-function falls into the exponential region on the left side of the curve. This is a region where Ei-function curve behaves more like exponential curve and losses the logarithmic behavior. As Feature Formulation 3 captured the logarithmic term

but not the exponential one, it failed to match the early time in our case. When t gets larger, the Ei-function moves to the logarithmic region on the right. This explains the good behavior using Feature Formulation 3 after early time has passed.

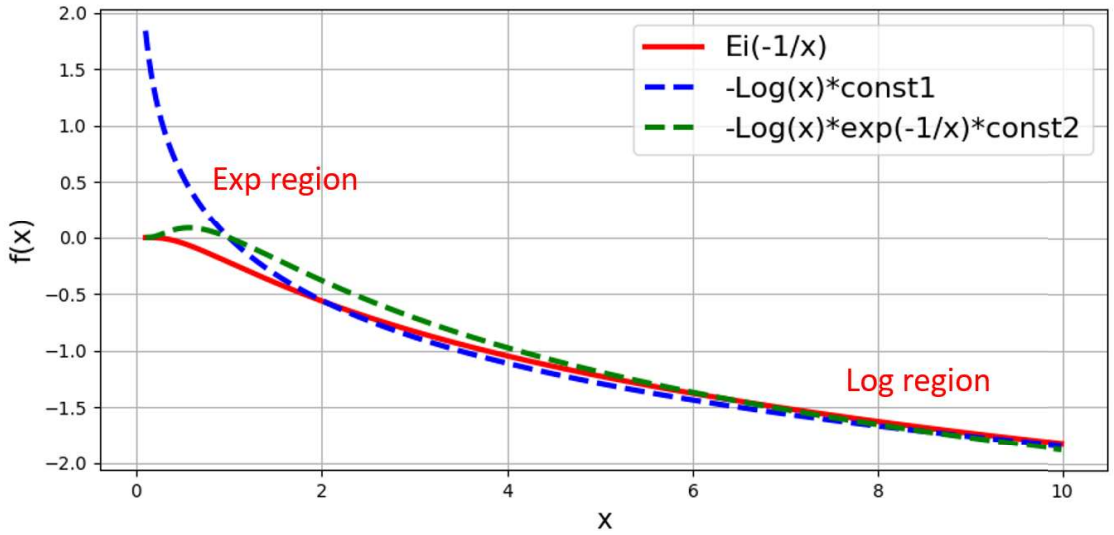

Figure 3.16: Comparison of: 1) $Ei(-\frac{1}{x})$, which governs the actual pressure-rate relationship, 2) $log(x)$ as in Feature Formulation 2 and 4, and 3) $\frac{log(x)}{exp(\frac{1}{x})}$ as in Feature Formulation 5. The term x here represents $\frac{1}{(t^{(i)}-t^{(j)})}$.

To solve the issue, we used Feature Formulation 4, but again, with careful tuning of parameter values inside Ei function. In this case, we used parameter values from 10^{-8} to 10^{-1}. The center figure in Fig. 3.15 shows that this formulation solved the problem and matched the actual data quite well.

An an alternative way, adding exponential terms in Feature Formulation 5 also eliminated the pressure jump problem, see bottom figure in Fig. 3.15. It was observed that Feature Formulation 4 and 5 produced very similar result. However, Feature Formulation 5 does not require Ei parameter tuning as in Feature Formulation 4, which is better from a practical standpoint.

To sum up this section, in Case 1, we saw earlier that Feature Formulations 3, 4, and 5 all performed well. Now in Case 2, Feature Formulation 4 and 5 worked well and Feature Formulation 3 produced the undesired pressure jump issue at the early time.

3.2.2 Algorithm Performance

In Case 2, the results were similar to the result of Case 1, in a sense that the algorithms that performed well in Case 1 also performed well in Case 2. In Fig. 3.17, the mean squared error scores of Support Vector Regression, Ridge Regression, SGD Regression, and Neural Network are lower

than the other algorithms.

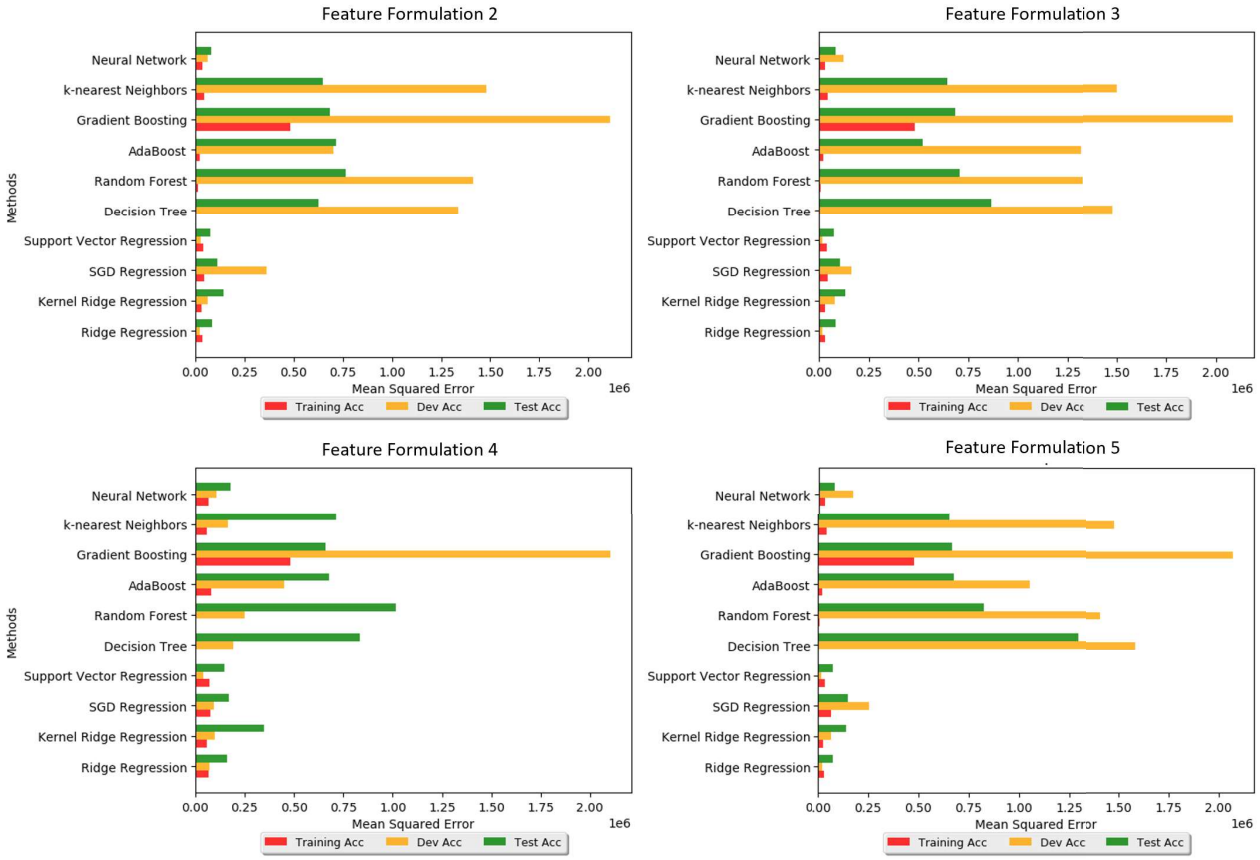

Figure 3.17: Mean squared errors comparison between different Feature Formulations (Case 2).

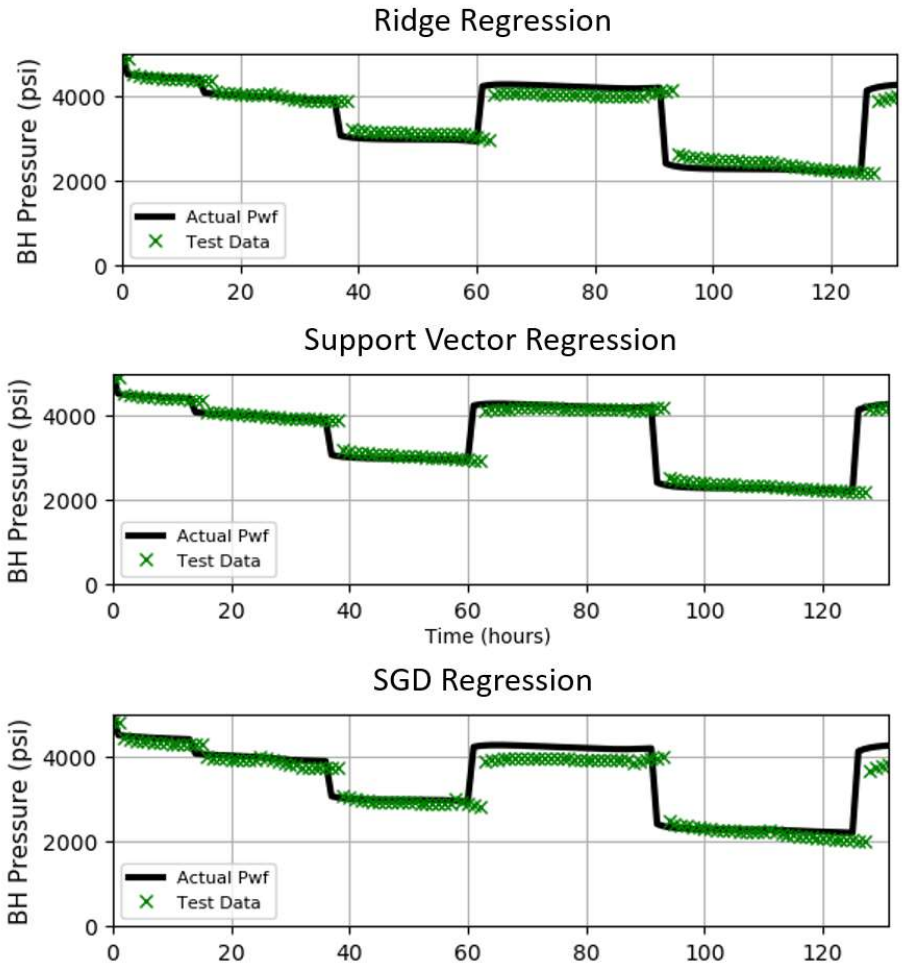

Figure 3.18: Resulting bottom hole pressure prediction using Ridge Regression, Support Vector Regression, and Stochastic Gradient Descent (SGD) Regression, all with Feature Formulation 4.

Support Vector Regression appeared to be the best performer, with the highest development and test scores, see Fig. 3.18. Ridge Regression came second in terms of development and test scores. SGD Regression generally captured the trend although there were small noises at certain places due to the stochastic nature of the method. The usefulness of SGD regression was not quite seen in this problem, as the size of dataset was small. SGD regression might be useful in a big data setting where performing batch update is computationally expensive. Neural Network also performed well using Feature Formulation 4 and 5. Again, similar to Case 1, using Neural Network with raw data input to roughly estimate bottom hole pressure (with high tolerance) might be fine, but that is not the case if our expectation is to obtain a good pressure derivative prediction result.

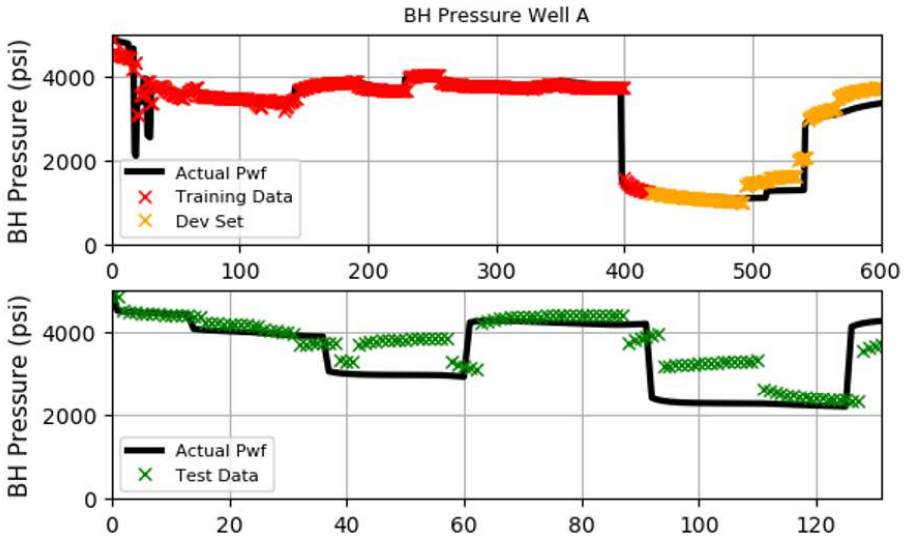

Figure 3.19: Resulting bottom hole pressure prediction using Kernel Ridge Regression with Feature Formulation 4 and polynomial kernel.

An interesting result can be observed in Kernel Ridge Regression. When performing hyperparameter search using Feature Formulation 4, the model chose polynomial kernel degree 2 as it gave the highest cross-validation score. In Fig. 3.19, the training set looked reasonable. However, when deploying the model in the test set, the performance was poor. The model mapped the features to higher dimensions and this over-complexity made the model more sensitive to small change in the feature. If the model was constrained such that only a linear kernel could be used, the performance was much worse.

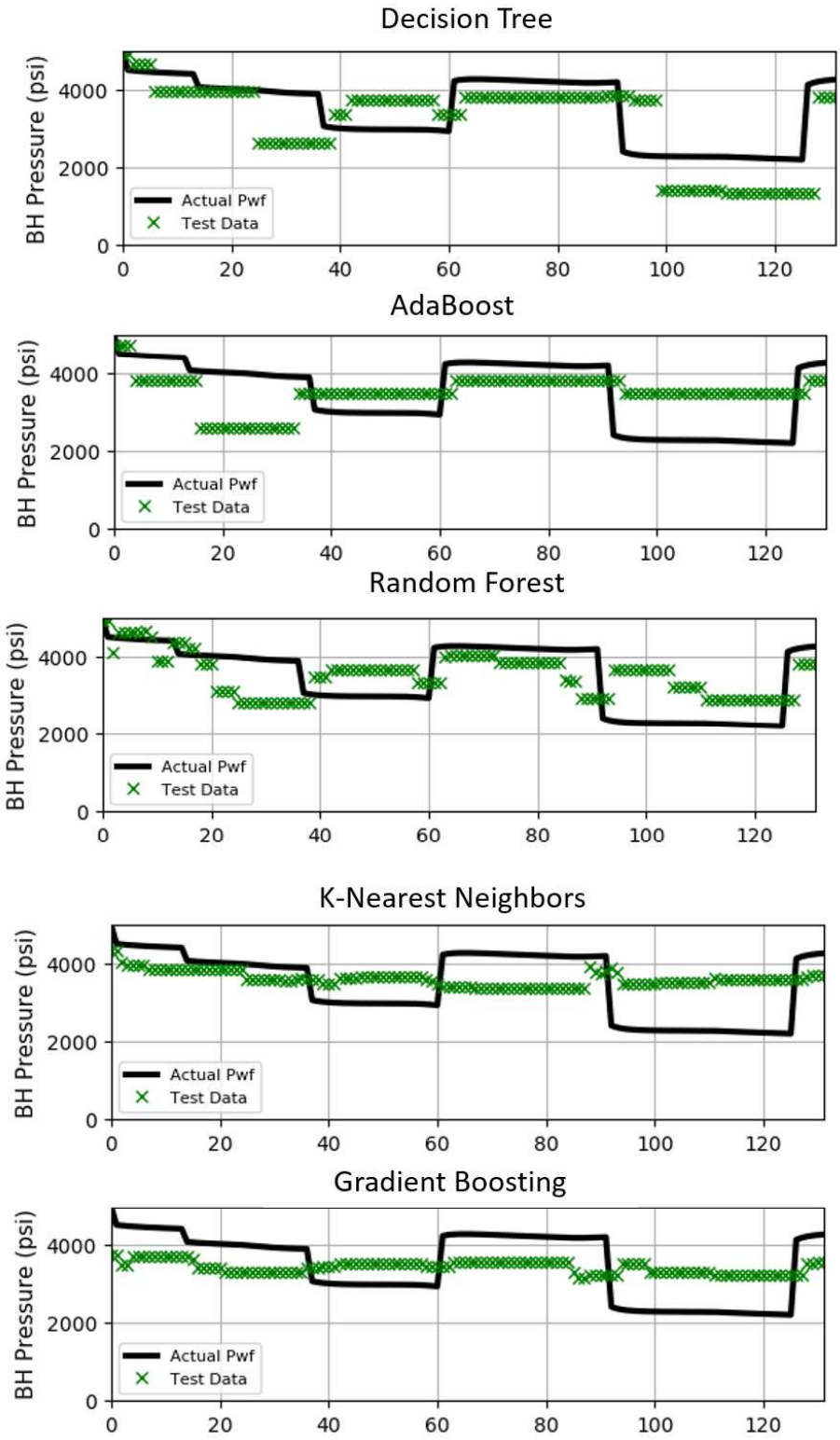

Figure 3.20: Resulting bottom hole pressure prediction using Decision Tree, AdaBoost, Random Forest, k-Nearest Neighbors, and Gradient Boosting, all with Feature Formulation 4.

Fig. 3.20 shows five algorithms that did not perform well in Case 2. These algorithms also scored low in Case 1. The issue was the same: overfitting. For that reason, these algorithms are not considered suitable for this time-series regression problem.

3.2.3 Noisy Data

So far, we have seen the machine learning application in clean (nonnoisy) dataset. In real world application, measured data are often contaminated with noise. The noise can be generated internally from the measurement device or it can come in from external sources, such as unstable flow.

We were interested to see the performance of all ten algorithms under noisy data and at what noise level the algorithms start to break. Artificial noise was added to the bottom hole pressure before the training the model. The noise had Gaussian distribution as formulated in Eq. 3.1.

$$p_{wf}^{(i)} = p_{wf}^{(i)} + \mathcal{N}(\overline{\mu} = 0, \sigma) p_{wf}^{(i)} \tag{3.1}$$

where σ (or standard deviation of the bottom hole pressure data) varied from 0 to 1. We performed the same procedure in Fig. 2.1 to obtain the predicted bottom hole pressure and R-squared scores. R-squared scores were calculated by comparing predicted bottom hole pressure and clean (nonnoisy) data. Feature Formulation 5 was used to preprocess the raw input.

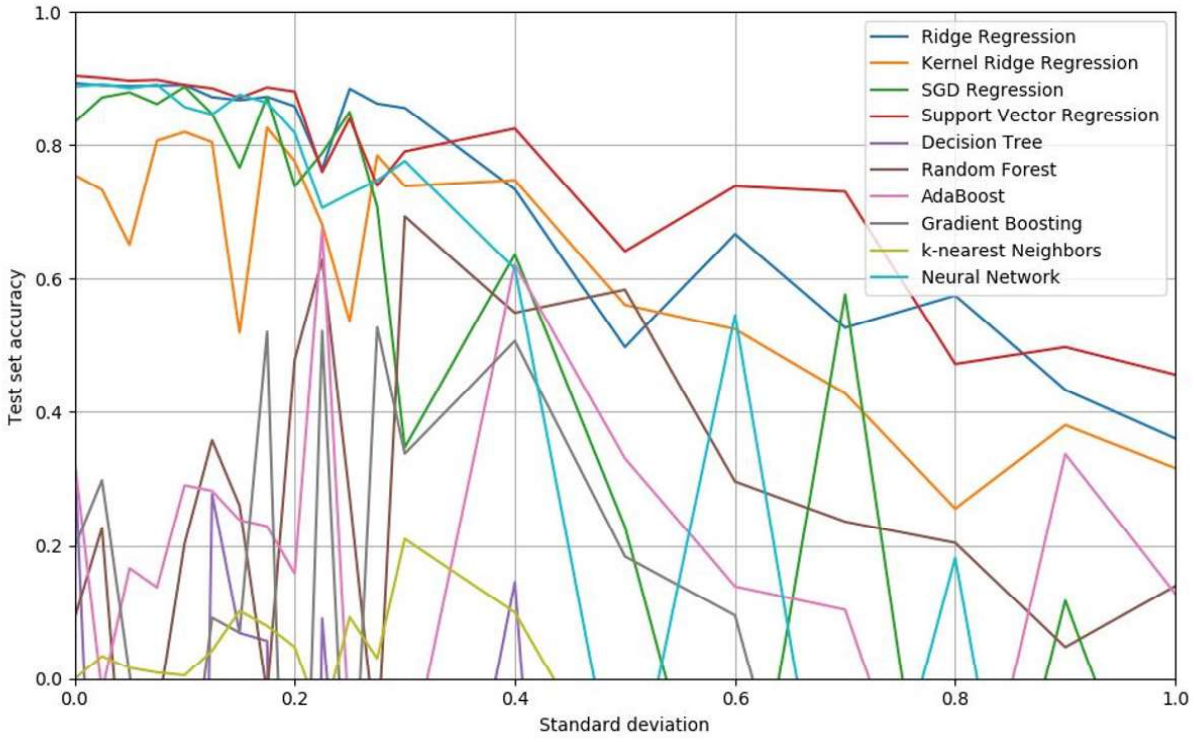

Figure 3.21: R-squared scores of ten different algorithms as a function of noise level.

Fig. 3.21 shows the comparison of R-squared scores in the test set. We saw earlier that Support Vector Regression, Ridge Regression, and Neural Network were the top-three algorithms with the highest development and test scores. As the noise level increases, it is expected that the test scores drop, however Kernel Ridge Regression and Regression remained the two best algorithms. At $\sigma = 0.3$, Neural Network score dropped rapidly and became unstable. The higher the noise level is, the harder it is to tune the Neural Network model to generalize well. Tree-based and ensemble algorithms did not perform well at any noise level.

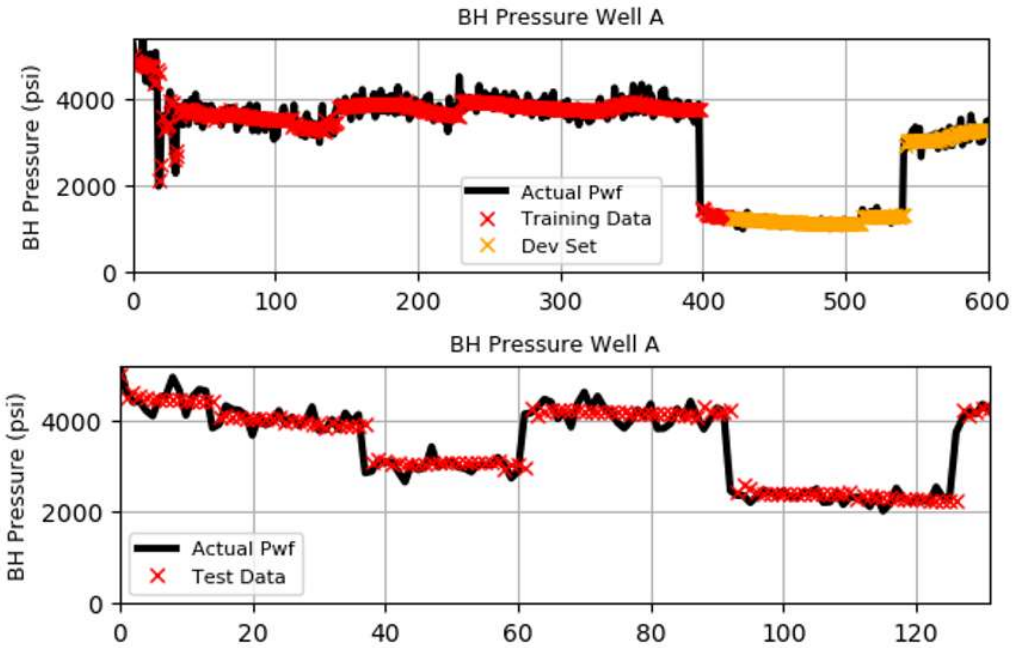

Figure 3.22: An example of a good result using Support Vector Regression at $\sigma = 0.05$. Top: training and dev set. Bottom: test set.

Fig. 3.22 shows the bottom hole pressure prediction result using Support Vector Regression. Although the bottom hole fluctuation is in the order of hundreds of psi, the machine learning model was still able to predict the actual signal (without noise). This was true as the Gaussian distribution was centered at the actual signal and some regression methods are natural to handle this type of noise. At $\sigma = 0.05$, Support Vector Regression scored 0.97 and 0.88 in the training and test sets, respectively.

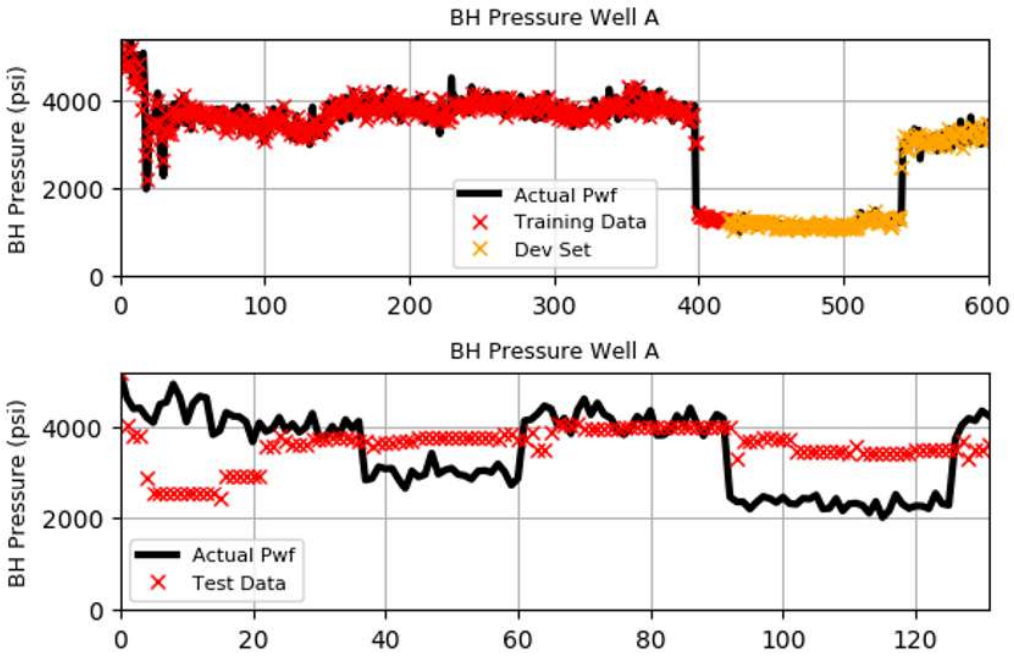

Figure 3.23: An example of a bad result using Decision Tree at $\sigma = 0.05$. Top: training and dev set. Bottom: test set.

Fig. 3.22 shows the resulting prediction using Decision Tree. The algorithm tried to fit each data point (including the noise) in the training set but did not generalize well in the test set. Decision Tree was hardly able to distinguish noise and actual signal. The same issue was also observed in Random Forest, AdaBoost, Gradient Boosting, and k-Nearest Neighbors.

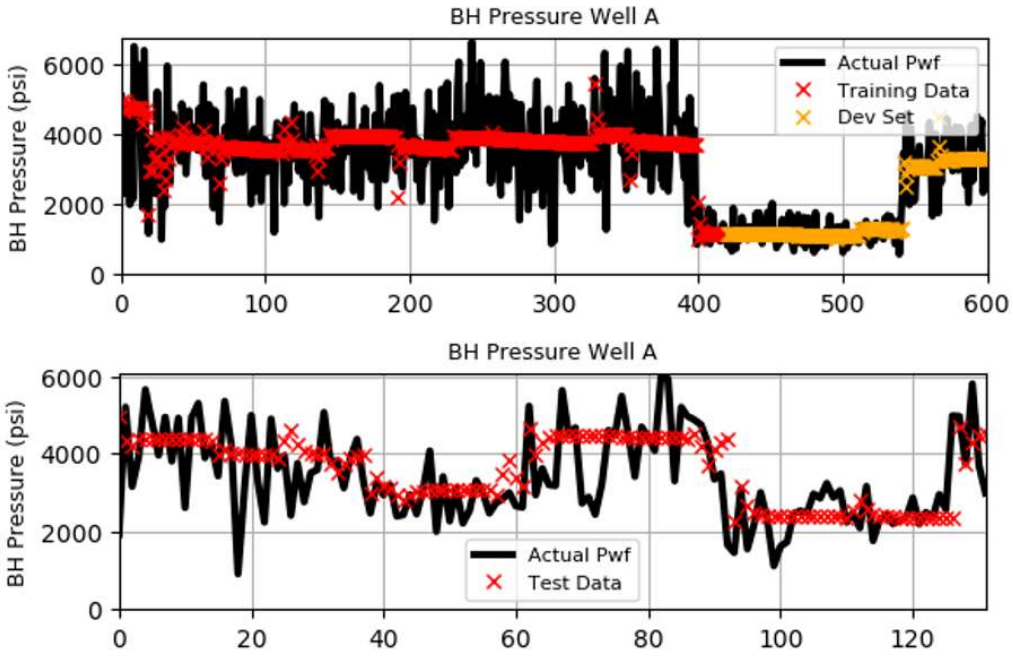

Figure 3.24: An example of a good result using Support Vector Regression at $\sigma = 0.25$. Top: training and dev set. Bottom: test set.

As the noise level goes up, it is increasingly harder to predict correct bottom hole pressure value, especially right after flow rate changes, see Fig. 3.24. After each flow rate change event, the model required enough data points before it was able to correctly guess the value of actual clean signal.

Chapter 4

Two-Phase Flow Rate and Water Cut Prediction

Flow rate reconstruction in a single-phase problem was studied previously by (Chuan 2015b). It was found that the convolution kernel method can deconvolve the pressure signal successfully without explicit breakpoint detection. The features were similar to Eq. 2.3, except that the term flow rate was changed to pressure and the fourth feature was removed, as shown in Eq. 4.1 below.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \end{bmatrix} \tag{4.1}$$

where i denotes the time step and j denotes the previous flow rate change events.

In the two-phase production problem, the bottom hole pressure is a response from both oil and water rate changes. As shown in Chapter 3, as long as oil and water rates are known, building a machine learning model that predicts the bottom hole pressure is doable. In contrast, using convolution of pressure to predict oil and water rates is trickier as it is harder to know which phase corresponds to a certain bottom hole pressure change.

In order to give the model more information about the flow rate state, we added total liquid rate as a feature in the input matrix (see Eq. 4.2). This may seem strange and one may question the point of predicting one of the flow rates given total (liquid) flow rate. A potential useful application might be when multiphase-flow-metering (MFM) system or separator test unit is not available all the time and only single-phase flow meter is permanently installed. This is common as MFM devices are more expensive than single-phase meters. There are cases, especially offshore, where deploying a

separator test unit is difficult and expensive. A machine learning model can be trained using the oil rate, water rate, and pressure data from temporary MFM system (or a separator test unit). When the MFM system is not in-place, we can use the pressure to predict individual flow rate (or water cut) of the well, assuming no flow-regime change in the wellbore.

$$X^{(i)} = \begin{bmatrix} \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right) \\ \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right) \log\left(t^{(i)} - t^{(j)}\right) \\ \sum_{j=1}^{i-1} \left(p_{wf}^{(j)} - p_{wf}^{(j-1)}\right)\left(t^{(i)} - t^{(j)}\right) \\ q_T^{(i)} \end{bmatrix} \tag{4.2}$$

where $q_T^{(i)}$ denotes total liquid rate (in this case oil + water rate).

4.1 Case 1

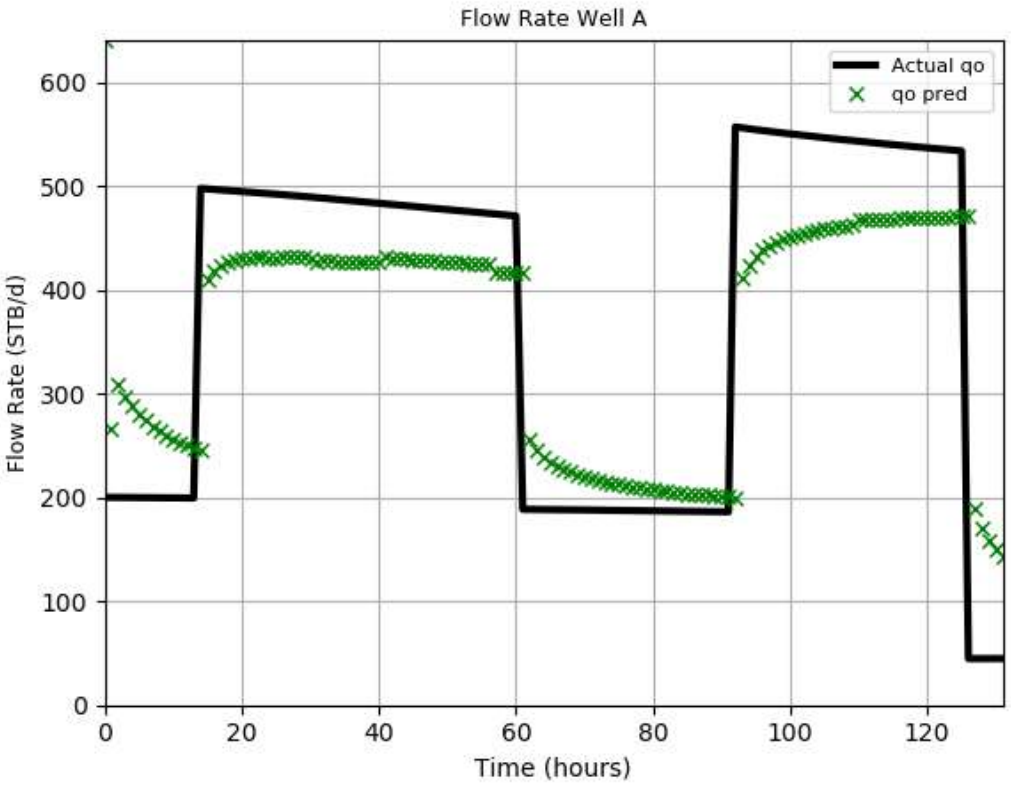

Figure 4.1: Oil rate prediction using Ridge Regression and input matrix in Eq. 4.1 (Case 1).

If we used Eq. 4.1 as the input and Ridge Regression as the model to predict oil rate, the resulting oil prediction rate was inaccurate by around 10-50% (roughly 10-80 STB/D) in the test set, as seen in Fig. 4.1. The high error was observed after every flow rate change event and it slowly converged to a certain flow rate value afterwards.

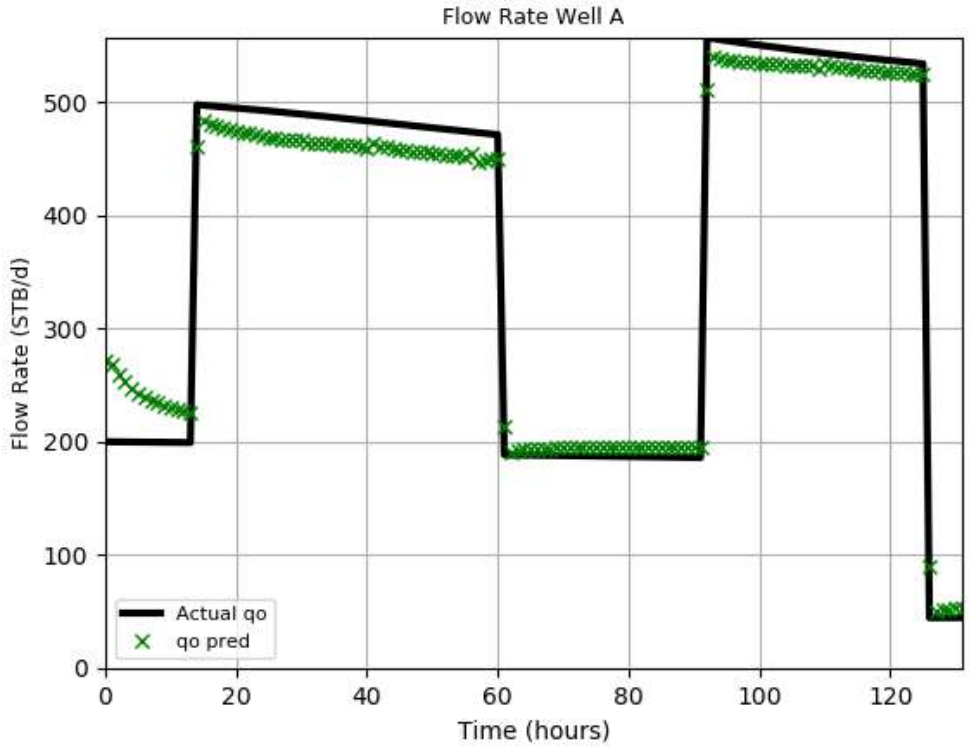

Figure 4.2: Oil rate prediction using Ridge Regression and input matrix in Eq. 4.2 (Case 1).

When we added total liquid rate as an input (as in Eq. 4.2), the performance was better (see Fig. 4.2). However, we still see incorrect prediction at early time at $t = 0$ to $t = 15$ hours. This was due to water cut change from 0 to 11% in the first 130 hours. The model was not able to take into account water cut evolution over time.

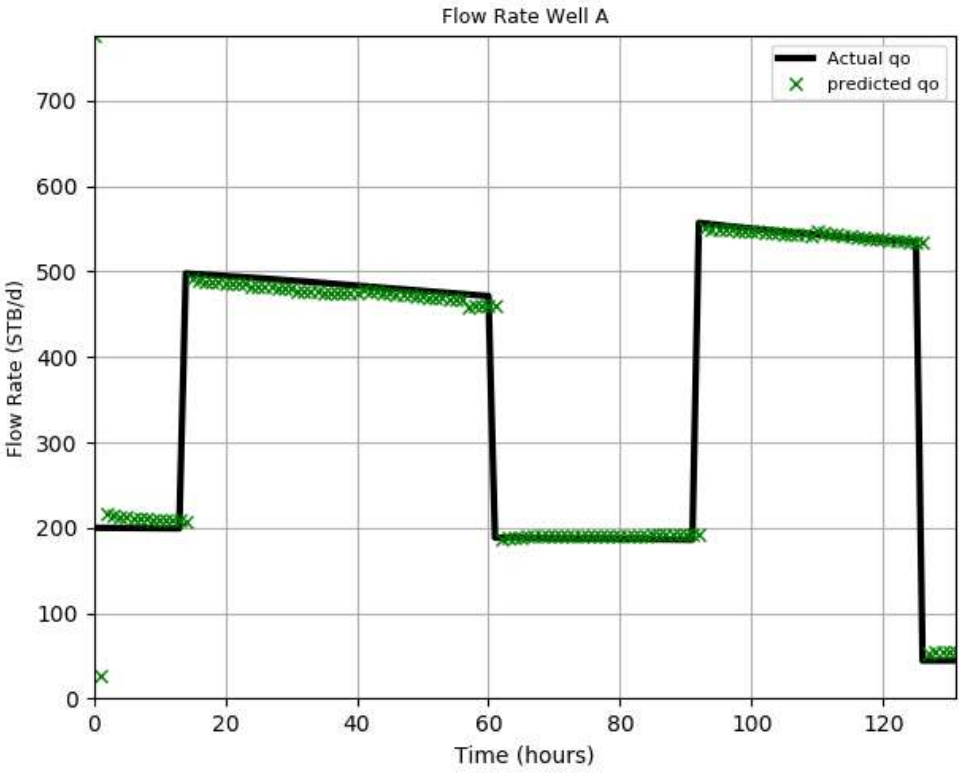

Figure 4.3: Oil rate prediction using Ridge Regression and input matrix in Eq. 4.2 (Case 1).

In Chapter 3, we saw that Support Vector Regression outperformed other algorithms in bottom hole pressure prediction problems. It turned out Support Vector Regression also performed well in oil rate prediction despite water cut variation over time, as seen in Fig. 4.3. SVR was able to achieve 0.76% of R-squared score in the test set and the oil rate prediction was only inaccurate by 0-10% (less than 20 STB/d).

Using the same feature formulation, we can also train a model that predicts the water rate. As water cut is just a fractional form of water rate compared to the volume of total liquid rate, the same feature formulation can also be applied. Fig. 4.4 shows the resulting water cut prediction. The prediction error grew over time and at the end of test period, we observed 5% difference in water cut. Building a machine learning model that is able to handle water cut variation requires further research. The water cut evolution post-breakthrough is complex. Using only flow rate and pressure to infer geometrical understanding of the well and reservoir as well as multiphase flow behavior is not an easy problem. Building a machine learning model that captures these things requires careful formulation and data selection.

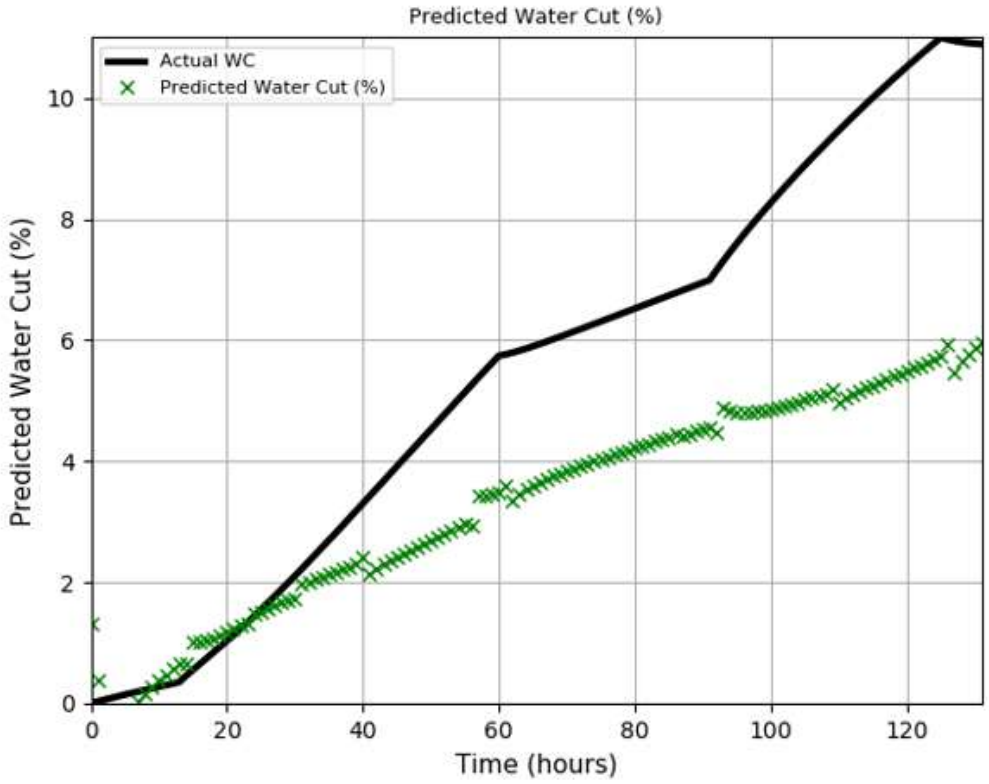

Figure 4.4: Water-cut prediction using Support Vector Regression and input matrix in Eq. 4.2 (Case 1).

4.2 Case 2

From Case 1, we learned that Support Vector Regression was reasonably good at predicting oil rate. Next, we applied similar procedure and formulation to the dataset in Case 2, where water cut change over time was relatively small.

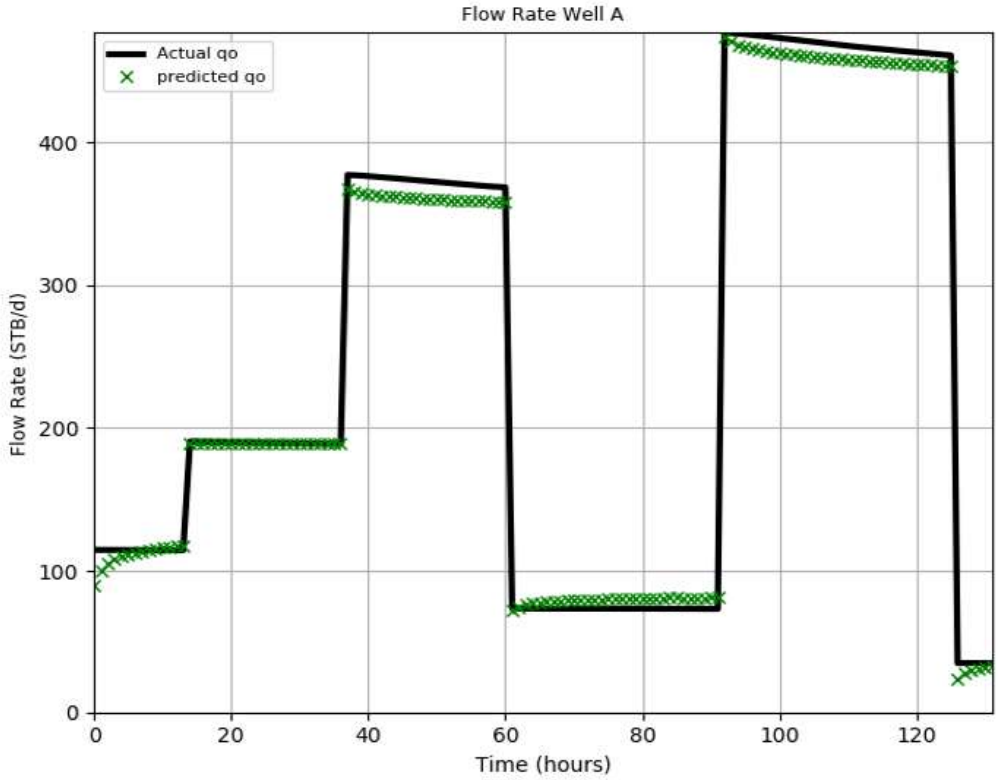

Figure 4.5: Oil rate prediction using Support Vector Regression and input matrix in Eq. 4.2 (Case 2).

Fig. 4.5 shows the resulting oil rate prediction. At the very beginning, the error was around 25% (25 STB/d difference), but after some time the model got better and yielded an acceptable difference less than 15 STB/d (<30%).

Case 2 was an easier problem for water-cut prediction as it only changed from 61.8% to 64.8% in 125 hours. The prediction result was not perfect, but was not poor either. The model captured the trend quite well with acceptable fluctuation of noise, see Fig. 4.6.

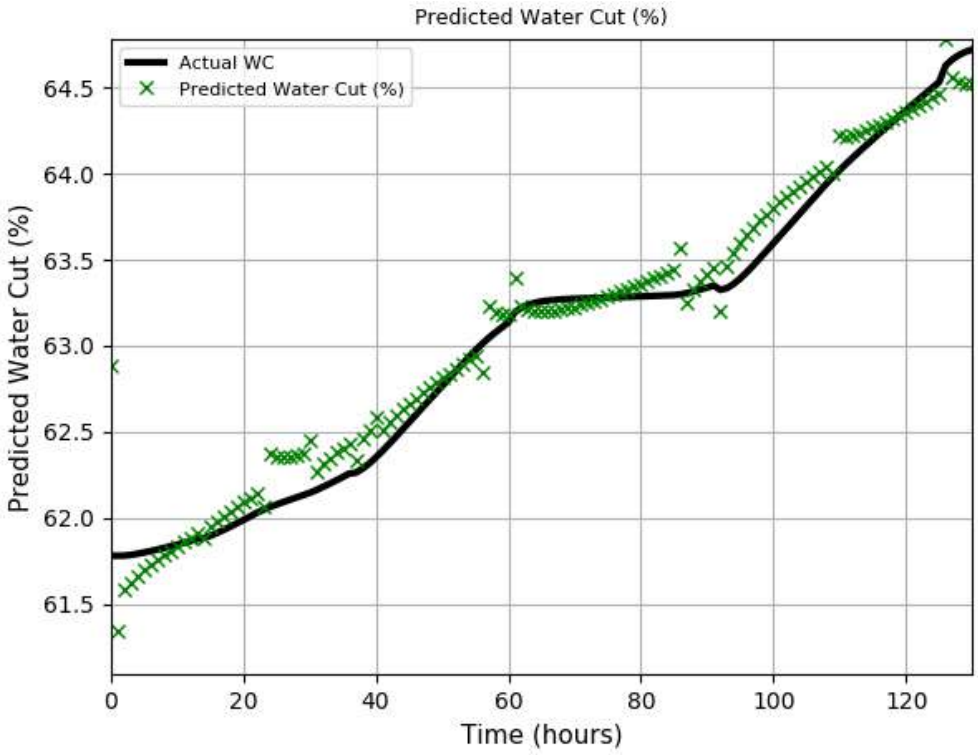

Figure 4.6: Water cut prediction using Support Vector Regression and input matrix in Eq. 4.2 (Case 2).

Chapter 5

Conclusions

5.1 Conclusions

In this study, we explored and implemented ten machine/deep learning algorithms for bottom hole pressure, oil rate, and water cut prediction in multiphase (oil and water) and multiwell production problems. Support Vector Regression and Ridge Regression generally performed well in two cases given the right features. On the other hand, tree-based algorithm, ensemble algorithms (AdaBoost, Random Forest, and Gradient Boosting), and instance based (k-Nearest Neighbors) did not perform well in our setting. The result is summarized as follows.

Table 5.1: Summary of qualitative algorithms performance, taking into account the MSE scores, R-squared scores, and derivative results.

Algorithms	Case 1	Case 2
Support Vector Regression	Good	Good
Ridge Regression	Good	Good
Fully-Connected Neural Network	Good	Good
Kernel Ridge Regression	Fair, with some fluctuations	Fair, with some fluctuations
Stochastic Gradient Descent Regression	Fair, with some fluctuations	Fair, with some fluctuations
Decision Tree	Poor	Poor
Random Forest	Poor	Poor
AdaBoost	Poor	Poor
Gradient Boosting	Poor	Poor
k-Nearest Neighbors	Poor	Poor

We also investigated the breaking limit of each algorithm given certain level of noise in the dataset. Among ten algorithms we investigated, Support Vector Regression and Ridge Regression topped in almost any noise level in our setting. Fully-connected Neural Network scored high on development and test sets, but the score dropped as the standard deviation of the noise went beyond 0.3.

In the oil rate prediction, we investigated the performance of Support Vector Regression and

Ridge Regression. Support Vector Regression performed very well and was able to handle water cut variation with acceptable difference, given the right feature formulation (either 4 or 5). Ridge Regression also generally performed well but had less ability to handle water cut variation than SVR. In the water cut prediction, both algorithms could predict small water cut variation (around 3 percent). However both algorithms failed when water cut changed more than 10 percent in the testing period.

Feature formulation strategy from previous work (Feature Formulation 3) in a single-phase flow problem was applied and this study found potential improvement in the early time behavior. We proposed two different feature formulations to improve the existing formulation. Feature Formulation 4 (see Eq. 2.6) and 5 (see Eq. 2.7) worked well in our setting. The result is shown in Table 5.1.

Table 5.2: Summary of qualitative feature formulations performance, taking into account the MSE scores, R-squared scores, and derivative results.

Feature Formulations	Case 1	Case 2
Feature Formulation 1	Poor	Poor
Feature Formulation 2	Fair	Fair
Feature Formulation 3	Good	Good in overall, but with some pressure jumps
Feature Formulation 4	Good	Good
Feature Formulation 5	Good	Good

5.2 Future Work

Further development is needed to develop a machine/deep learning model that is robust with changing water-cut over time. One question to answer is whether pressure and flow rate are sufficient to model the complex post-water breakthrough behavior. If the answer is no, it would be interesting to see what other parameters are required to build a good model.

Another potential model improvement is the robustness to changing reservoir pressure and well productivity index. The model used in this study was large enough such that reservoir pressure drop in the training and testing is small. This study also assumed constant productivity index. Theoretically, recurrent neural network should be able to handle the issue by passing the productivity index change information forward to the next time step. But this still requires an experimental proof.

This research assumed homogeneous porosity and permeability. But in reality, almost no homogeneous permeability or isotropic reservoir exists. The rock properties are learned and stored in the machine/learning model as weights or parameters. In a heterogeneous reservoir, as long as the reservoir properties do not change over time, the model should be able to capture the complexity of the heterogeneous reservoir in the parameters, as in the homogeneous case. However, this requires more investigation. The research can also be extended to address pressure-flow rate relationship in unconventional reservoirs.

A typical oil and gas reservoir produces oil, water, and gas phases. Building a machine/deep

learning model for three-phase problems would be more complicated, as gas phase exhibits different flow behaviors than water and oil at various pressure conditions.

In real oil and gas fields, it is common that a downhole pressure gauge and a flowmeter have different interval sampling. For instance, downhole pressure gauge records pressure every second and flow rate is measured every day (or even weeks). One can address this by preprocessing the data before entering into the machine/deep learning model and performing normal one-to-one mapping. Another alternative is to utilize a sequence model in deep learning to perform many-to-many mapping. This type of model has been used widely in machine language translation, which allows inputs and outputs to have different number of words.

Not all wells have the luxury of having downhole pressure gauge and only rely on surface/tubing head pressure. From a practical standpoint, it would be useful to build a model that takes tubing head pressure as an input instead of bottom hole pressure. This may require additional features as the model has to take into account flow regime and other phenomena inside the wellbore.

Nomenclature

α	Regularization strength control
μ	Viscosity
$\overline{\mu}$	Mean of input feature
ϕ	Porosity
σ	Standard deviation of input feature
θ	Parameter/weight
B	Formation volume factor
B	Number of classifiers in Random Forest
b	bias vector
C	Tolerance penalty control hyperparameter in SVR
c_t	Total compressibility
H	AdaBoost mapping function
h	Reservoir thickness
h_m	Number of classifiers in Gradient Boosting
h_t	Individual Tree function in Random Forest
h_t	Individual weak classifier in AdaBoost
K	Kernel matrix
k	Permeability
L	Loss function

p_i	Reservoir pressure
p_{wf}	Bottom hole pressure
q_o	Oil flow rate
q_T	Total liquid rate
q_w	Water flow rate
r	Well distance
T	Number of classifiers in AdaBoost
T	Number of classifiers in Gradient Boosting
t	Time
W	Weight matrix
X	Feature matrix
X'	Standardized feature matrix
y	Target/ground truth vector

Bibliography

[1] Liu, Y. and Horne, R.N., (2013), "Interpreting Pressure and Flow Rate Data from Permanent Downhole Gauges Using Data Mining Approaches", PhD Thesis, Stanford University.

[2] Tian, C. and Horne, R.N., (2015a). "Applying Machine Learning and Data Mining Techniques to Interpret Flow Rate, Pressure, and Temperature Data from Permanent Downhole Gauges", SPE Western Regional Meeting.

[3] Horne, R.N., (1995). "Modern Well Test Analysis", Petroway, Palo Alto, CA, second edition.

[4] Hastie, T. et al., (2001). The Elements of Statistical Learning. Springer New York Inc., New York, NY, USA.

[5] Tian, C. and Horne, R.N., (2015b). "Machine Learning Applied to Multiwell Test Analysis and Flow Rate Reconstruction", Society of Petroleum Engineers ATCE.

[6] Tian, C. and Horne, R.N. (2017). Recurrent Neural Networks for Permanent Downhole Gauge Data Analysis. SPE ATCE.

[7] Tian, C. (2014). Applying Machine Learning and Data Mining Techniques to Interpret Flow Rate, Pressure and Temperature Data from Permanent Downhole Gauges. MS report.

[8] Handscomb, S., Sharabura, S., and Woxholt, J., (2016). "The Oil and Gas Organization of the Future". McKinsey. [Online]. Available: https://www.mckinsey.com/industries/oil-and-gas/our-insights/the-oil-and-gas-organization-of-the-future.

[9] Pedregosa, F., Varoquaux, G., Gramfort, A. & Michel, V. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[10] Ng, A., (2017). Machine Learning (CS229) Course Notes. Stanford University.

[11] Li, F., Johnson, J., Yeung, S., (2017). Convolutional Networks for Visual Recognition (CS231N) Course Website. Stanford University. (Available online: http://cs231n.github.io/)

[12] Friedman, J.H. (2011). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics. JSTOR.

[13] Zell, Andreas (1994). "chapter 5.2". Simulation Neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.). Addison-Wesley.

[14] Shawe-Taylor, J., & Cristianini, N. (2004). Kernel Methods for Pattern Analysis. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511809682

[15] Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984). Classification and Regression Trees. Belmont: Wadsworth.

[16] Boomer, R.J. (1995). Predicting Production Using a Neural Network (Artificial Intelligence Beats Human Intelligence). Society of Petroleum Engineers (SPE 30202).

[17] Suhag, A., Ranjith, R., and Aminzadeh, F. (2017). Comparison of Shale Oil Production Forecasting using Empirical Methods and Artificial Neural Networks. University of Southern California. SPE ATCE (SPE-187112-MS).

[18] Grus, J. (2015). Data Science from Scratch. Sebastopol, CA: O'Reilly. pp. 99, 100. ISBN 978-1-491-90142-7.

[19] Smola, J., Scholkopf, B. (2003), "Tutorial on Support Vector Regression". NeuroCOLT Technical Report TR-98-030.

[20] James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013, "An Introduction to Statistical Learning", Springer, New York.

[21] Drucker, H., (1997). Improving Regressors using Boosting Techniques. In ICML (Vol. 97, pp. 107-115).

[22] Shrestha, D. L., & Solomatine, D. P. (2006). Experiments with AdaBoost. RT, an improved boosting scheme for regression. Neural computation, 18(7), 1678-1710.

[23] Glorot, X., & Bengio, Y. (2010, March). Understanding the Difficulty of Training Deep Feedforward Neural Networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256).

[24] Shanno, David F. (July 1970), "Conditioning of quasi-Newton methods for function minimization", Mathematics of Computation, 24 (111): 647–656, doi:10.1090/S0025-5718-1970-0274029-X, MR 0274029