University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Automated Well Monitoring: Machine Learning and Web Application

Master's Thesis in Computer Science
by

Anisa Zhurda

Internal Supervisor

Prof. Chunming Rong

External Supervisor

Dr. Anton Shchipanov
NORCE

June 15, 2020

Abstract

The challenge within the oil and gas industry is that of complexity and therefore cost, specifically due to the tough working environments and delays/downtime [1]. Therefore, digitization is proposed as a cost saving opportunity ,making data collection through sensors and data analytic approaches priority in the industry. The integration of machine learning has already shown its contribution in augmenting human decision making and optimizing processes. However deciding the right technique keeps being a challenge for further reducing the cost. In this thesis, we are assembling several machine learning and deep learning models and testing them with the aim of optimally reconstructing missing flow rates in well monitoring data. The experiments are focused in some simple regression models such as Linear Regression, Ridge Regression, Kernel Ridge Regression and Gradient Boosting Regression and one deep learning model for time-series: LSTM. Except of its original form, we are applying two feature engineering techniques on the well data: convolution method and transient reduction method. The experimental results shows the importance of feature transformation in performance of the models by emphasizing two moments: the dramatic improvement of Kernel Ridge Regression over the convolutional transformed data and the outstanding ability of LSTM to learn on raw data. We finalize our work by creating a simple web application for reconstructing the missing flow rate values using the most optimal machine learning model, by giving a chance for everyone to reuse and interact with the model.

Acknowledgements

I am very grateful to my supervisor Prof. Chunming Rong from University of Stavanger who trusted me to work on this thesis.

I would like to express my special gratitude to Dr. Anton Shchipanov from NORCE for providing me his guidance, ideas, encouragement and insightful feedbacks amidst his busy schedule.

Contents

Chapter 1

Introduction

1.1 Motivation

The oil and gas industry is rapidly changing by emerging into more dynamic and complex environments, as it is facing constant challenges in terms of supply and demand.[2] As this industry continues to adapt to a sustained period of volatility, companies are taking concrete steps by adopting digital technologies with the aim to increase efficiency and safety,improve production processes and reduce cost.

At the center of all these digitization efforts stands an exciting technological field, presumably dominating major industries and life-sectors: Artificial Intelligence (AI). Primary applications of this technology, within the oil and gas industry, are attained via advanced machine learning (ML) and deep learning (DL) techniques.

In 1959, Arthur Samuel defined machine learning as a "Field of study that gives computers the ability to learn without being explicitly programmed".[3] With massive amounts of computational power, machines are now able to analyze large sets of data observations and apply relationship modeling in an inferential and predictive way, in real time and with high accuracy. Machine learning and other big data applications could save the oil and gas industry as much as $50 billion in the coming decade, according to McKinsey. [4]

Reservoir being the core of the entire operation in upstream sector, needs to be maintained and optimized to increase the longevity of the whole production cycle. Therefore, sensors such as Permanent Downhole Gauges (PDGs) are installed permanently in a well to provide continuous record of pressure,temperature and sometimes also flow rate from the use of flow meters during operations. The fundamental idea is to show that machine learning has the potential to handle the complexities in well data analysis and find patterns behind the data,which will implicitly contain the well and reservoir information.

In this thesis,we are addressing two main problems which are significant to well data analysis ,namely, reconstruction of missing flow rate values by applying ML and DL techniques and website development for user interaction with the most optimal predictive model achieved in the first task. As inferred, presence of such gaps in flow rate records is still a usual case in the petroleum industry due to installation of flow-meters for clusters of wells and following rate allocation per well. Therefore we are testing and comparing different models,to reproduce or predict well flow rate time series based on pressure time series from the PDG data. Then the predictive model will be presented through an interactive web application,where the user uploads the corresponding well data with missing values of flow rate and get a reconstructed version resulted by a data-driven approach. Results will be interpreted in graphical,tabular form and as a csv file the user can download for further usage.

1.2 Related work

1.2.1 Feature-Based machine Learning for PDG Data Analysis

Feature-based machine learning aims to identify the patterns behind PDG measurements, for instance, the relationship between flow rate and pressure data. Such patterns contain the reservoir information implicitly, and can be utilized in various ways. One use is to identify the reservoir model with pressure deconvolution by feeding a simple rate history into the trained flow rate-pressure relationship. Another use is to predict the reservoir performance with rate control as inputs to the trained patterns. [5]

The first work of this kind was initiated by Liu and Horne (2012, 2013a, 2013b) [6] [7] [8]. They introduced the convolutional kernel approach to model the pressure convolution effects, and validated the method on several synthetic and real cases. The subsequent work by Tian and Horne (2014)[9] addressed some of the issues in Liu and Horne's work, including early transient behaviors and computational efficiency.

Tian and Horne (2015)[10] also extended the applications of feature-based machine learning to multiwell testing, flow rate reconstruction and interwell connectivity problems. Being a topic of interest in our thesis, we are analyzing a bit further the approach to flow rate reconstruction presented in this paper. After modeling new features using the pressure measurements from PDG data, the previously proposed convolutional kernel approach was tested on both synthetic and real dataset. A high agreement of true values and predictions were observed in the synthetic case (fig 1.2/a) although the unseen data was quite different from the one used for training the model. Moreover ,promising results

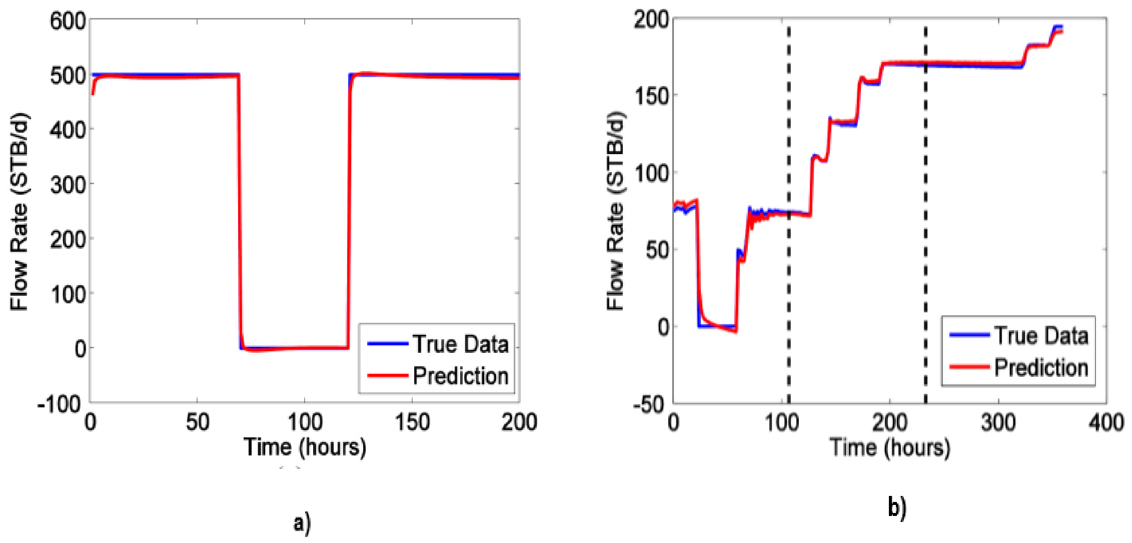


Figure 1.1: Reconstructed flow rate in synthetic (a) and real data (b) by Tian and Horne 2015b

(fig 1.2/b) were also shown from the real-life usage of this method, demonstrating a flexibility of machine learning in adapting new set of features and learning from them.

All those works relied on the careful handcrafting of features, which are identified as candidate factors to model the patterns based on knowledge of the physics. An inappropriate feature design can hurt the modeling results significantly. Thus it becomes challenging to apply feature-based machine learning when the understanding about the physics is limited,that means a forward model is not available.

1.2.2 Deep Learning for PDG Data Analysis

Although some of the core concepts in deep learning such as back-propagation date back to the 1980s,it has been only in recent years that deep learning has risen in various research topics and engineering problems. Deep learning is also often referred to as deep neural networks, which reveals the technical foundation on which deep learning is built. One key characteristic of deep learning is that it does not require the laborious feature handcrafting process but learns from the raw data directly. In fact, the information about features are captured by the unique structure of deep neural networks.

Until now, there have been a very limited number of studies of deep learning in reservoir characterization. Aggarwal et al. (2014)[11] used NARX to conduct a simulated well testing, but considered only one synthetic test case with simple model settings. Korjani et al. (2016)[12] studied how deep learning can help estimate petrophysical characteristics

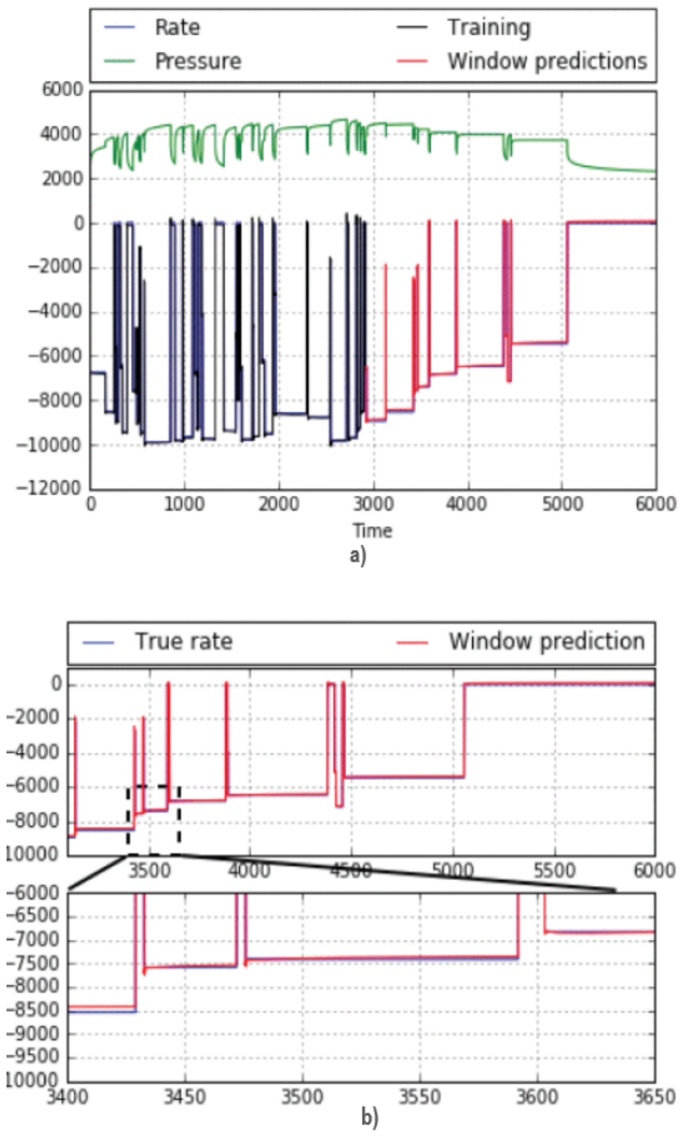


Figure 1.2: LSTM predictions of flow rate in synthetic dataset (a),shown also in a zoomed interval (b)by Heghedus et al. (2019)

by generating synthetic logs of newly drilled wells, and reported that deep learning outperformed kriging for that case.

Heghedus et al. (2019)[13] compared NARX and LSTM to forecast pressure and rate using a synthetic dataset of PDG data and flow meters, driven by the success deep learning approaches have shown in the energy consumption field. The main contribution of this paper was the adaptation of shifting window method in well data, to boost the performance of LSTM specifically in flow rate prediction. The results shown by LSTM further extended the applicability of Deep Learning in all the industries operated with wells.

In this thesis we will be referring the concept of convolutional kernel technique proposed by Tian and Horne (2015b) and the LSTM model applied by Heghedus et. al (2019) to reconstruct missing flow rate. Our aim is to create a wider perception of feature modeling and machine learning techniques applicable to well data and examine differences among them.

Chapter 2

Problem statement

2.1 Dataset

Two datasets with well measurements of flow rate and pressure over time ,were simulated on a reservoir model and used in this study. The first dataset is considerably small, with observations measured hourly for approximately 8 months.For each pressure transient there is a constant flow rate record represented as a step function in time, which as seen from Table 2.1 has negative value. Moreover,we observe that in consecutive pressure transients,if the pressure values are increasing the flow rate is decreasing ,indicating a negative relation .

	Elapsed time(hr)	Water rate	Pressure
0	0.018177	-6742.395145	2149.773050
1	0.024235	-6742.395145	2168.851204
2	0.030294	-6742.395145	2184.962183
3	0.036353	-6742.395145	2199.005331
4	0.042412	-6742.395145	2211.508740

Table 2.1: Sample of first dataset

Different from the first dataset, the measurements for the second one were separated in different files accordingly [time , pressure] and [time , flow rate]. Pressure is recorded more frequently, so to combine the data together over time, the following steps were taken:

1. Find the start and end time for each pressure transient
2. If there is a value of rate recorded in a specific time that falls in this range ,assign the rate to observations inside that time period.

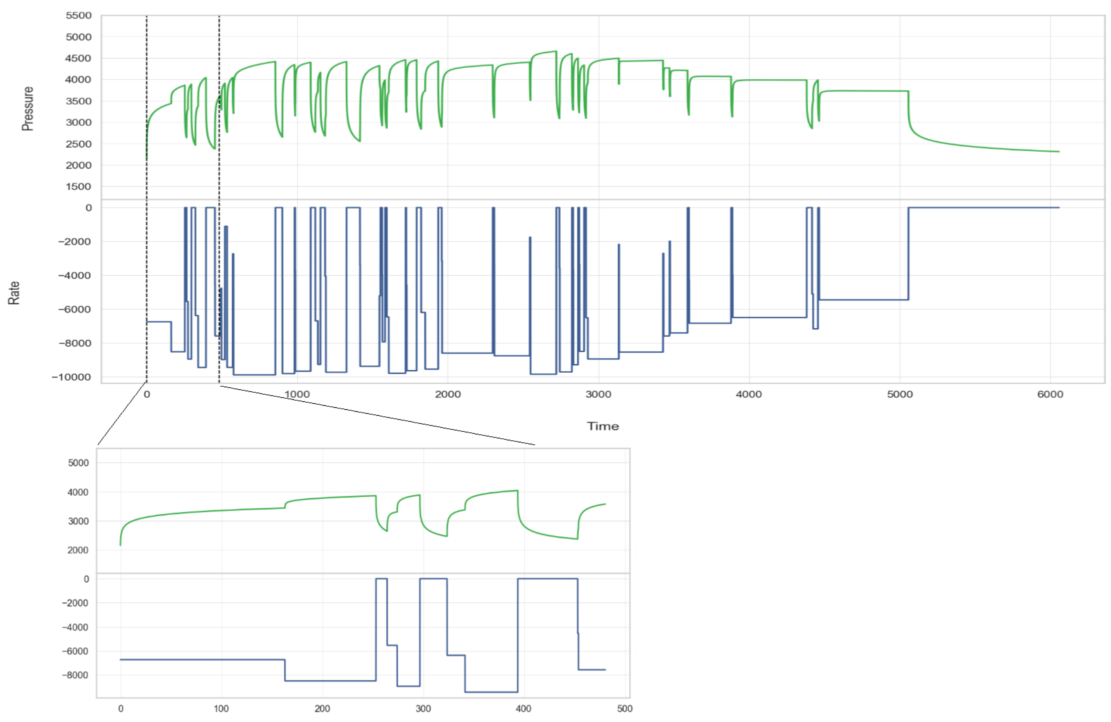


Figure 2.1: Graph from first dataset

As a result, a similar dataset with parameters elapsed time, flow rate and pressure is created ,for a period of approximately 8 years. Despite the positive values of flow rate, the relation between the pressure transients and flow rate remains the same (Fig 1.2). For the sake of comparison , only a small part of the second dataset is presented in the graph.

Due to the fact that the dataset is synthetic and clean, noise will be excluded as a possible issue while handling the data. A complication appears when a continuous section of flow rate is missing over the total time period during a pressure transient. In order to reconstruct the missing values of flow rate, several machine learning models will be built with target flow rate and features selected by different interpretations of time and pressure measurements.

2.2 Data preparation

Since feature selection has a huge impact on model's performance ,it is considered to be on of the most important steps. Three techniques were applied in this thesis:

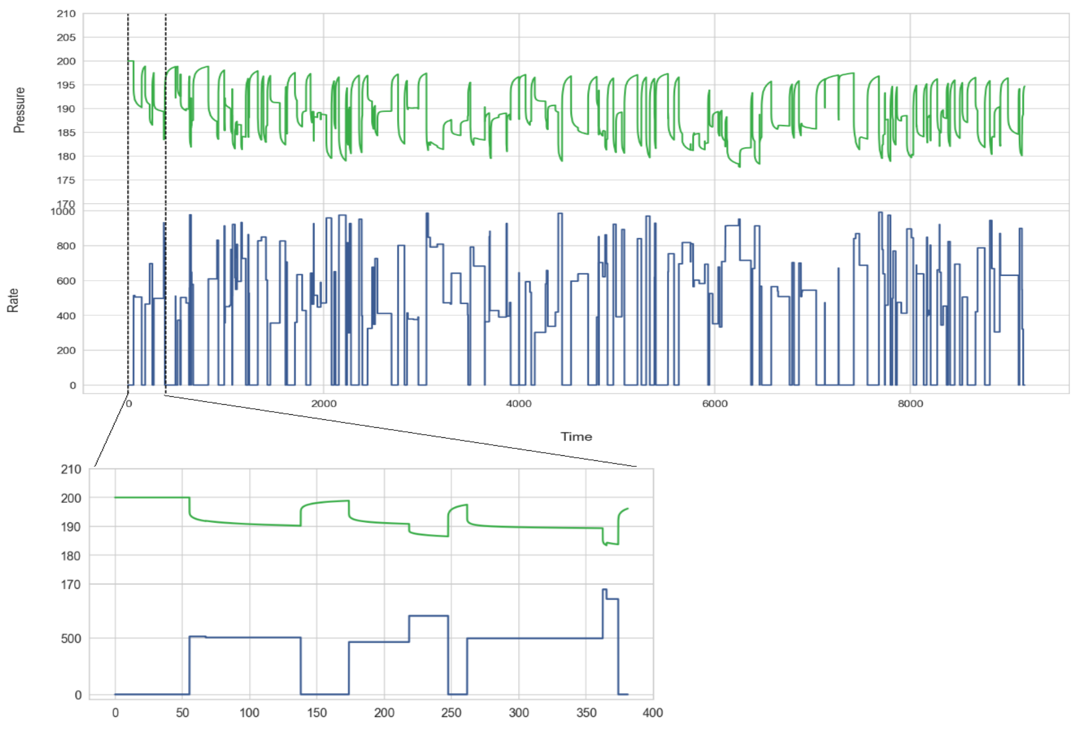


Figure 2.2: Graph from second dataset.

1. Using the behaviour of pressure over time as feature, without further changes, in order to find a dependency with flow rate $q(i)$.

$$x^{(i)} = \begin{bmatrix} t^{(i)} \\ p^{(i)} \end{bmatrix}, i = 0, ..., n$$

2. Exploiting the relationship between pressure changes and flow rate over one pressure transient. For each of the transients, we calculate delta pressure, starting time, duration time and corresponding flow rate. This will shrink the total number of observations in the dataset to the number of the transients (m).

$$x^{(k)} = \begin{bmatrix} t^{(k-1)} \\ t^{(k)} - t^{(k-1)} \\ p^{(k)} - p^{(k-1)} \end{bmatrix}, k = 1, ..., m$$

So, the target is $q^{(k)}$, which is the constant flow rate for each transient.

3. Based on the convolution approach that was proposed by Tian and Horne [10]. The vector of features where p stands for pressure and t time, is represented as:

$$x^{(i)} = \begin{bmatrix} \sum_{j=1}^{n}(p^{(j)} - p^{(j-1)}) \\ \sum_{j=1}^{n}(p^{(j)} - p^{(j-1)}) \log(t^{(i)} - t^{(j-1)}) \\ \sum_{j=1}^{n}(p^{(j)} - p^{(j-1)})(t^{(i)} - t^{(j-1)}) \end{bmatrix}, i = 1, ..., n$$

   Thus, the target flow rate is $q^{(i)}$ for n observations.

We observe from Table 2.1 that the data features vary highly in range ,which will affect in the contribution of features to the prediction. In order to make each feature similarly significant, we have to apply feature scaling. MinMaxScaler is an estimator from scikit-learn library that scales and translates each feature individually in a given range ,between 0 and 1.

After normalization, we split the data to train and test set by following the usual approach of 80/20 split intuition.

2.3 Predictive modeling

Depending on the performed task, there are two machine learning algorithms :supervised and unsupervised. In supervised learning we have both input variables (p,t) and an output variable (q) and we are using an algorithm to learn the mapping function from input to output. If the outcome is quantitative, then this is classified as a regression problem. On the other hand, the target can be categorical and the problem is classified as a classification problem. In unsupervised learning, the outcome is not provided and the main goal is to find similar patterns among observations. Due to the nature of our target,this study is focused on regression problem .

2.3.1 Evaluation metrics

Several machine learning models are chosen to be evaluated over each of the feature sets extracted by the PDG data and flow meters. Before discussing the models details ,we will focus on four evaluation techniques that will handle the performance of models in terms of error rate and explained variance . Since this is a regression task, we are using the following evaluation metrics :

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|q^{(i)} - \hat{q}^{(i)}|$$

- Root Mean Squared Error (RMSE)

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

- Mean Absolute Percentage Error (MAPE)

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|q^{(i)} - \hat{q}^{(i)}|}{q^{(i)}} \times 100\%$$

- R^2 or Coefficient of Determination.

$$R^2 = 1 - \frac{MSE(model)}{MSE(baseline)}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(q^{(i)} - \hat{q}^{(i)})^2$$

  where q is the flow rate in n observations ,and the baseline MSE is the mean over the target data.

Generally, the best model is the one that shows low error rate (MAE, RMSE, MAPE) and high determination coefficient (R^2). However, we will further explain the behavior for each of these metrics while we analyze the results and include the computational cost in terms of model fitting time.

2.3.2 Model details

Linear Regression

We start with the simplest machine learning model, in order to explain any tracks of linearity between features and target: Linear Regression. This model predicts the target as a weighted sum of the feature inputs, providing an easy interpretation on modular level, given by the equation:

$$q^{(i)} = \theta^T x^{(i)}$$

where q is the target flow rate and x the features, for observations i=1,..,n.

The “best” model is defined in terms of minimizing the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\theta^T x^{(i)} - q^{(i)})^2$$

The value of θ that minimizes $J(\theta)$ is given in a closed form (Hastie et al., 2009):

$$\theta = (X^T X)^{-1} X^T q$$

Although it looks simple, the linear regression approach shows a high level of interpretability. Since all the features are related directly to the target, the impact of each feature could be detected easily. As feature expansion is applied, the model becomes less interpretable. This will also make the learning procedure more like a trial-and-error, because we lose the guidance from the physical essence. However, as data distribution is changed to include more complex reservoir behaviors, linear regression fails to show a high learning ability and the kernel method comes in to give the algorithm more flexibility in describing reservoir characteristics.

Kernel Method

Linear regression allows the algorithm to learn in the feature space whose dimension is defined by X. There are certain situations which require mapping of the features X to higher dimension features $\varphi(X)$.

The kernel helps the algorithm to learn in $\varphi(X)$ dimensional space without explicit representation of $\varphi(X)$. This property allows us to add more features into our model with little cost, thus gives the model more flexibility to capture more detailed reservoir behaviors. The form of kernel should be carefully chosen to give an appropriate number of feature dimensions according to the complexity of the problem. The equation from Linear Regression can now be rewritten as :

$$q^{(i)} = \sum_{j=1}^{n}\beta_j K(x^{(i)}, x^{(j)}), i = 1, .., n$$

However, adding more features into the model can result in too much complexity and cause overfitting. To handle this issue we will introduce some useful tools to regularize the model so as to reduce the variance of predictions.

Model Regularization

Model regularization is a technique that shrinks the parameter estimates (in our case,θ and β). Model regularization is widely used to address the overfitting issue by reducing the prediction variance (the reason will be shown later in this section). Generally there are two ways of model regularization: ridge regression and lasso.(James et al.,2013), but we will be focused only on ridge regression. Starting from linear regression, we can shrink the linear coefficients adding penalty to the cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\theta^T x^{(i)} - q^{(i)})^2 + \lambda\sum_{j=1}^{p}{\theta_j}^2$$

where parameters θ that minimize the cost function are calculated as:

$$\theta = (X^T X + \lambda I)^{-1} X^T q$$

and $\lambda \geq 0$ is known as the tuning parameter. Choosing the right value for λ is quite challenging because it is a bias-variance trade off problem.

Ridge regression can also be applied on the kernel framework, such as Kernel Ridge Regression, when we want to achieve the feature expansion . The cost function will be presented as :

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{n}\beta_j K(x^{(i)}, x^{(j)}) - q^{(i)}\right)^2 + \lambda\sum_{j=1}^{p}{\beta_j}^2$$

The solution of parameters β is given by (Shawe-Taylor and Cristianini, 2004) with a defined kernel matrix K_M:

$$\beta = (K_M + \lambda I)^{-1} q$$

Overall, ridge regression is particularly useful to address the overfitting problem when used with the kernel approach as we will later see from the results. [9]

Regression Tree

Then we introduce a tree ensemble learner,such as Gradient Boosting, to test how regression trees will handle this forecasting task.

"Boosting" in machine learning is a way of combining multiple simple models into a single composite model. This is also why boosting is known as an additive model, since simple models (also known as weak learners) are added one at a time, while keeping existing trees in the model unchanged. As we combine more and more simple models, the complete final model becomes a stronger predictor. [14]

The term "gradient" in "gradient boosting" comes from the fact that the algorithm uses gradient descent to minimize the loss. By using gradient descent, predictions are updated consistently according to a learning rate (α) ,with the goal of minimizing the loss function (MSE).

LSTM

After testing the performance of these traditional machine learning models, a remarkable deep learning model proposed by Hochreiter and Schmidhuber in 1997 is applied to the time-series data : Long short-term memory (LSTM).

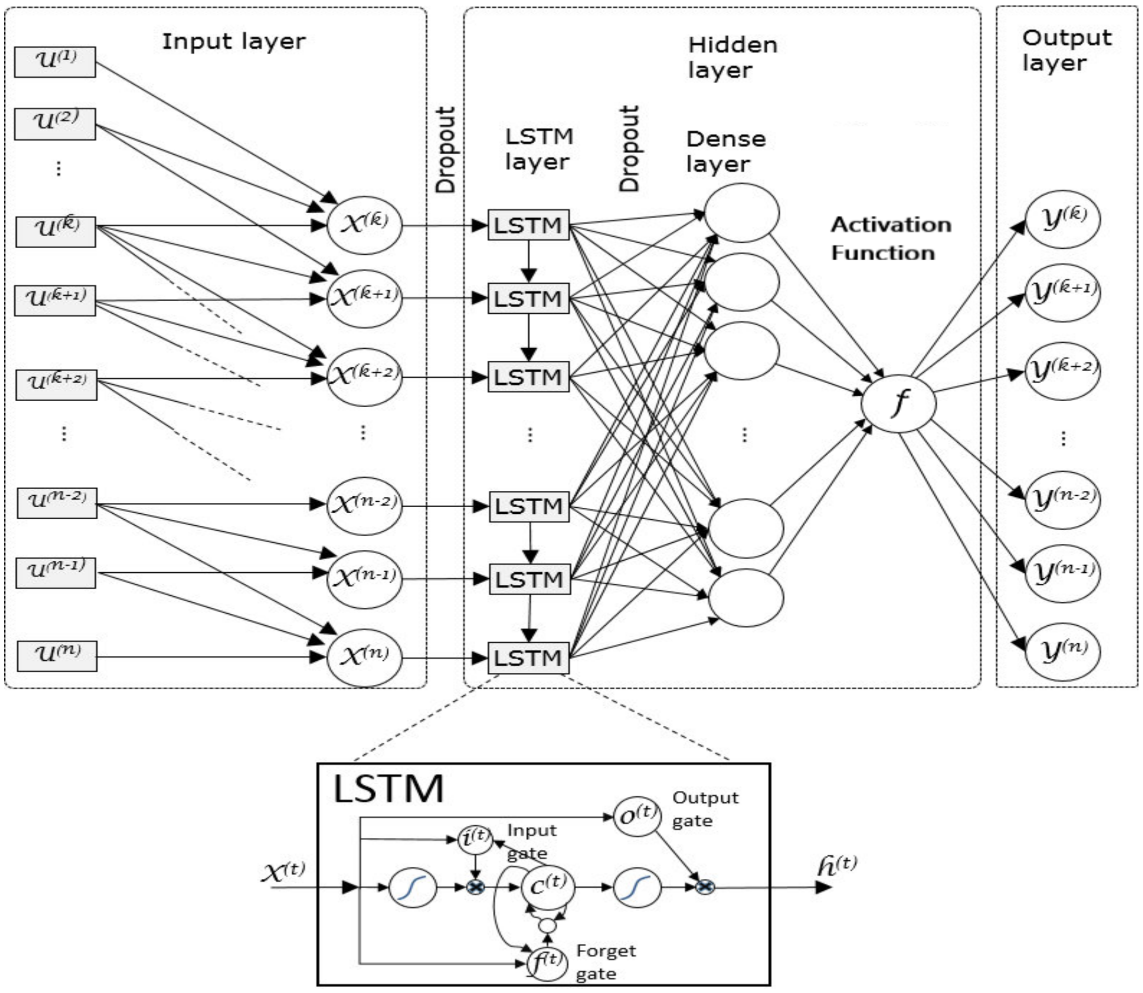


Figure 2.3: RNN with LSTM deep learning architecture

A really good representation of RNN with LSTM deep learning architecture is given by George Loukas et al. (2017) [15] in figure 2.3. In terms of the deep learning architecture,the design consists of three main layers:

1. Input Layer: Time-series data corresponds in n data observations which are grouped as k consecutive points to increase the detection latency.
2. Hidden Layer: It includes a Long-Short Term Memory (LSTM) layer, a Dense layer and Activation Function. Each LSTM neuron consists of input gate,forget gate and output gate which determine the significance of the input and whether it should continue to remember its value or forget it, and when it should output it. This is followed by a Dropout Layer where the number of hidden nodes serve as the main tuning parameter, and activation function introduces non-linearity into the output of a neuron.
3. Output Layer: After receiving the output the error between the actual value and the predicted value is calculated (MSE).Then we will backpropagate through time,to update the weights and LSTM cell states to optimize the results.

The selective memory function of LSTM makes it suitable to deal with time sequence prediction problems such as our well data. A better understanding of the mathematical computations in LSTM can be achieved from the paper LSTM: A Search Space Odyssey [16]

Chapter 3

Experimental Evaluation

3.1 Experimental Setup

To the extent of the model analysis in our regression task, the following main python libraries will assist in building the predictive models:

1. Scikit-learn, a library built upon the SciPy (Scientific Python) ,is focused on modelling data and it provides a range of supervised and unsupervised learning algorithms.[17] In this thesis we are using Python version 3.6.10 and Scikit-learn version 0.22.1. As inferred, this library is used to build the following regression models: Linear regression, Ridge Regression,Kernel Ridge Regression and Gradient Boosting.

2. Tensorflow,a library for fast numerical computing, can be used to create Deep Learning models directly. On top of Tensorflow ,we are capable of running Keras which is a high-level neural networks API, written in Python.[18] For compatibility among libraries , we have chosen to work with Tensorflow version 2.1.0 and Keras 2.3.1. Hence we are building a Keras LSTM model and evaluate the deep learning approach to our problem.

3.2 Experimental Results

In this section we will analyze the data-driven results of several simple regression models and a deep learning model, based on three feature transformation techniques applied to our dataset, as demonstrated in the diagram in fig. 3.1.

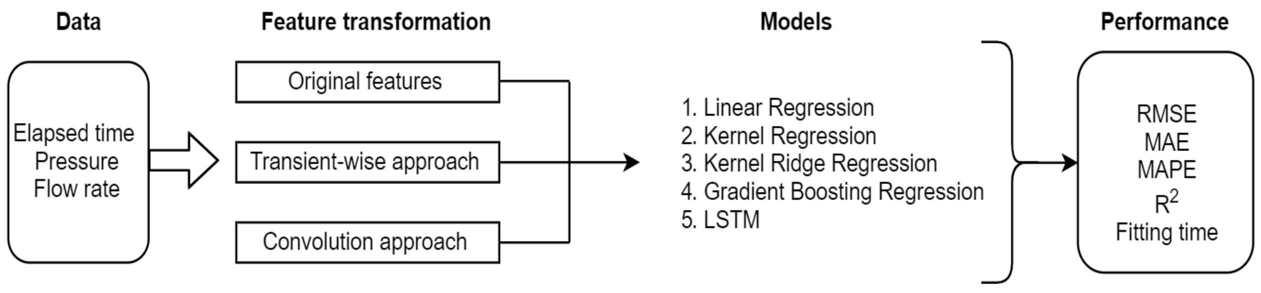


Figure 3.1: Experiment workflow

The dataset transformed by the convolution method has shown very promising results,so we will start by interpreting it in details and then proceed with a brief explanation of the other methods.

3.2.1 Feature transformation: Convolution approach

The initial data measured by PDG and flow meters, shown in table 2.1, will be transformed based on the convolution method that was previously introduced as a feature transformation method. The following dataset is achieved .

	Feature 1	Feature 2	Feature 3	Water Rate
0	-6.704426	-3.151023	-11.138749	514.791
1	-6.777953	-3.955461	-12.623886	514.791
2	-6.853799	-4.777462	-14.308884	514.791
3	-6.931966	-5.618032	-16.221045	514.791
4	-7.012430	-6.478161	-18.391429	514.791

Table 3.1: Convolution approach: Data transformation

Model	Train				Test				Fitting Time(s)
	RMSE	MAE	MAPE	R^2	RMSE	MAE	MAPE	R^2	
Linear Regression	0.0907	0.032	2.252	0.931	0.107	0.064	4.868	0.912	0.002
Ridge Regression	0.0902	0.030	2.135	0.932	0.106	0.060	4.605	0.918	0.23
Kernel Ridge Regression	0.006	0.002	0.148	0.9981	0.0021	0.0012	0.105	0.9988	29.87
Gradient Boosting Regressor	0.007	0.005	0.187	0.999	0.107	0.039	2.689	0.934	9.54

Table 3.2: Convolution approach :Evaluation results for simple regression models

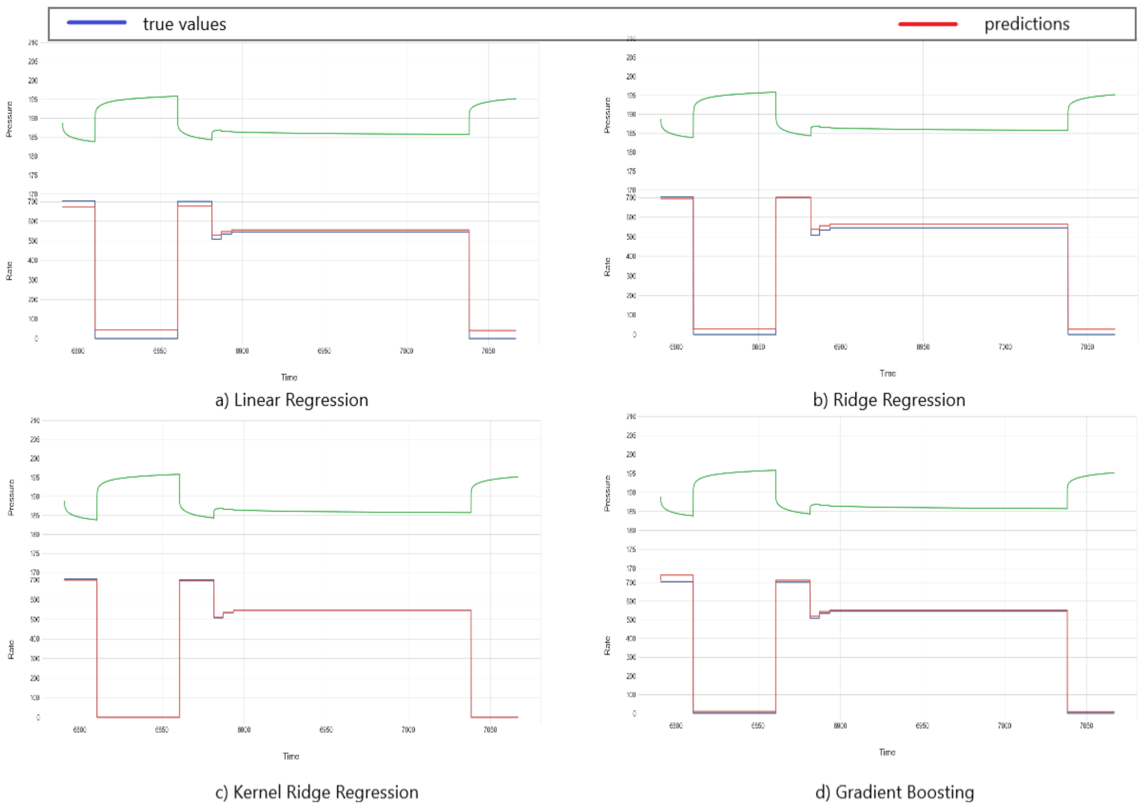


Figure 3.2: Convolution approach: Predictions of simple regression models

Linear regression implementation is quite straightforward, using LinearRegression() class from scikit-learn library, as no additional parameters are required in the process. We will fit the model by applying the training dataset, with 3 features and flow rate as target ,which will explicitly calculate the intercept and feature coefficients. As we can see from the results in figure 3.2, 93% of the variance in flow rate is described from the corresponding three features in the training set and 91.2% in the test set, with quite a small error rate. These results show a good performance of linear regression compared to its later applications.

For Ridge regression, we are performing grid search by tuning the parameter alpha. We start from 0 where the model behaves identically like the simple linear regression to larger alpha values as we introduce higher smoothness constraint. The optimal alpha is determined to be 6. The model shows better error rate and a slightly better R^2 value.

Kernel Ridge Regression shows exceptional performance ,with the highest explained variance 99.8% and a significant drop in error rate for both training and test sets. Moreover the prediction in unseen data is slightly better than the train prediction. The optimal parameters that gave these results are :

$$\left\| alpha = 0.01, gamma = 2000, kernel =' rbf' \right\|$$

- Small value of alpha improves the conditioning of the problem and reduces the variance of the estimates.
- Radial basis function kernel (kernel="rbf") with the parameter gamma, which defines how far the influence of a single training example reaches. In our case high values refers to "close".

Lastly, Gradient Boosting Regressor is implemented as we tune the parameters to achieve a good trade-off of learning rate and number of boosting stages to perform, correspondingly 0.5 and 1000. Even though the train results show a promising performance similar to Kernel Ridge, this model tends to overfit with a poor prediction in unseen data. However, in comparison with Linear Regression and Ridge Regression it shows a slightly better performance.

In addition to evaluation metrics, the graphical representation of the test predictions zoomed in a small interval for each of the machine learning models in figure 3.2, shows that Kernel Ridge regression is certainly the most optimal regression model to be used in this specific task. However, fitting this model takes more time which can be computationally expensive.

On the other hand, in order to test the deep learning approach on the transformed well data, a Keras LSTM model is built. Prior to building the model, two essential steps are taken to modify the dataset to be compatible with the model:

- Creating lagged data with a range $(t, t + 5)$, shifting by 1 for each observation, which will lead to an expansion of the features from 3 to 19.
- Since the model will expect the input component to have the dimensions [samples, timesteps, features], we will reshape both training and test data into this new 3-dimensional structure. Correspondingly, the timesteps are chosen to be 2 in this experiment.

Regarding the model, we followed a trial and error approach to get the best results, which leaves space for further improvements. In this experiment, we define a model with different LSTM units in the hidden layer and an output layer that predicts a single numerical value: the flow rate . Every LSTM layer should be accompanied by a dropout layer which reduces the sensitivity to the specific weights of individual neurons: 30%

LSTM model	Train				Test				Fitting time(s)
	RMSE	MAE	MAPE	R^2	RMSE	MAE	MAPE	R^2	
20 nodes	0.0226	0.0116	0.736	0.9956	0.0232	0.0106	0.657	0.9957	418.57
50 nodes	0.0160	0.0080	0.515	0.9978	0.0182	0.0088	0.562	0.9973	723.53
100 nodes	0.0165	0.0086	0.526	0.9976	0.0150	0.0062	0.381	0.9982	890.50
200 nodes	0.0159	0.0059	0.367	0.9978	0.0152	0.0059	0.367	0.9982	1114.32

Table 3.3: Convolution approach: Evaluation results for LSTM with different number of hidden nodes

dropout rate is a good compromise in terms of accuracy and overfitting. The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or 'mse' loss function. As an activation function we have chosen relu which is computationally efficient and non-linear , and an L2 kernel regularizer to further reduce overfitting. As a general rule of thumb : 1 hidden layer works good enough to find reasonably complex features. The experiments are done with 500 iterations and a batch size of 10.

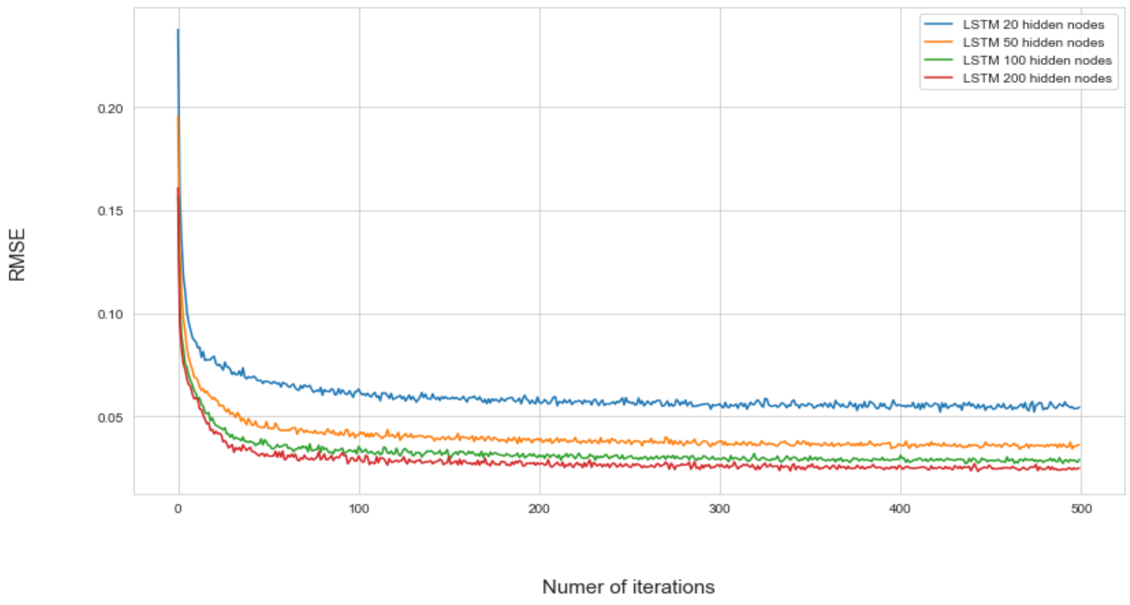


Figure 3.3: Convolution approach: RMSE for LSTM with different number of hidden nodes

We can infer from the results that the measured error is dropping with the increase of the number of hidden nodes. Hence the LSTM model with 200 hidden nodes is performing better. However the dropping rate starts to decrease for a large number of nodes. As we can see from the table 3.3 and graph 3.3, the lines representing RMSE are getting closer to each other with quite a small difference in error rate. Furthermore, the complexity of

LSTM model is demonstrated by the amount of time it takes for the model to fit, which is positively correlated to the number of hidden nodes applied.

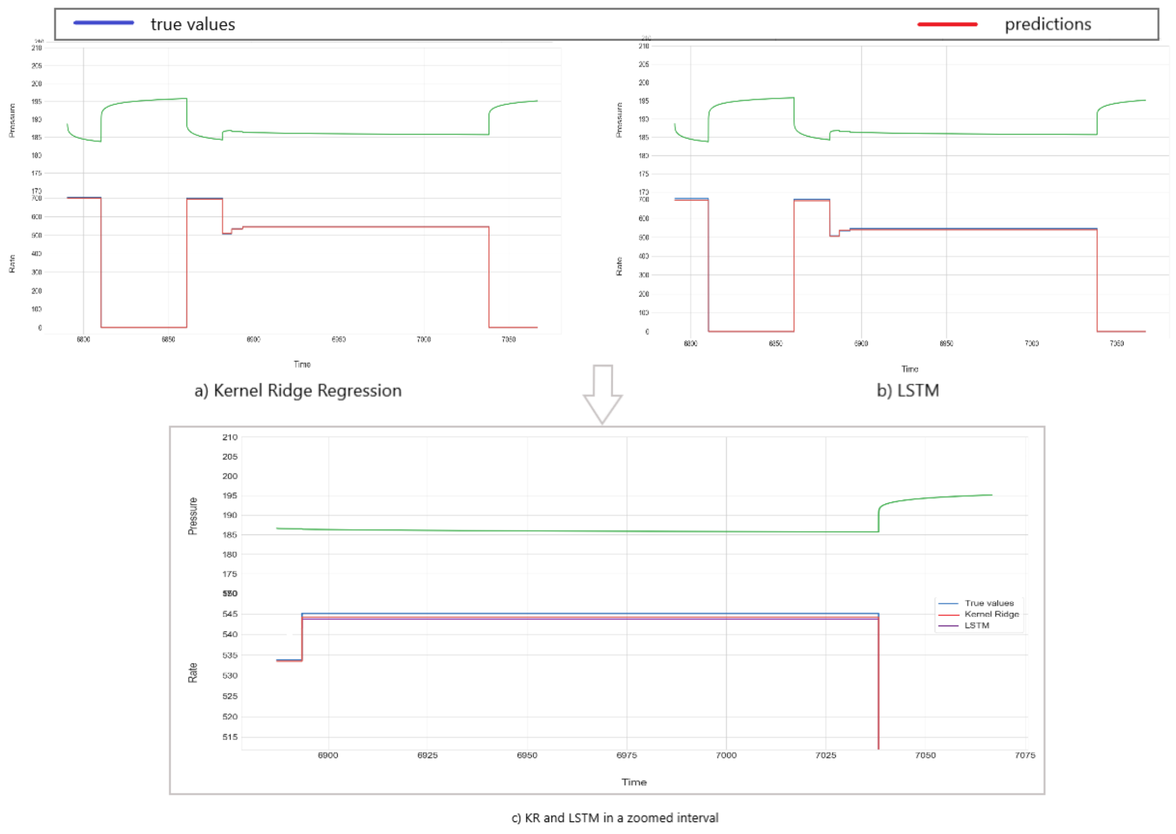


Figure 3.4: Convolution approach: Kernel Ridge and LSTM with 200 hidden nodes

The predictions from LSTM model are quite accurate (fig 3.4), but this experiment indicates a better performance of Kernel Ridge Regression which yields the most accurate predictions.

3.2.2 Feature transformation: Transient-wise approach & Original Features

If we closely observe the prediction graphs for both Kernel Ridge Regression and LSTM in the previous experiment, we notice that the predicted flow rate values for one transient vary slightly. Since our goal is to reconstruct a single value of flow rate per pressure transient, we can further smooth that by getting the median of values in one transient or we can follow another feature transformation technique based on transients. The new dataset will look as following:

Another advantage of this method stands in the reduction of the size of observations by making it more computationally efficient to learn from this data. However,this is

	Start time	Duration	Delta pressure	Water Rate
0	0.000	55.315	0.000	0.000
1	55.325	12.026	-4.046	514.791
2	67.361	70.617	-1.680	505.615
3	137.988	35.706	4.614	0.000
4	173.704	44.762	-4.294	465.312

Table 3.4: Transient-wise approach: Data transformation

	Train				Test				
Model + FT	RMSE	MAE	MAPE	R^2	RMSE	MAE	MAPE	R^2	Fitting + FT time(s)
Kernel Ridge Convolution Approach	0.006	0.002	0.148	0.997	0.0021	0.0012	0.105	0.998	29.87 + 7893.8
Gradient Boosting Transient-Wise Approach	0.105	0.065	3.591	0.897	0.126	0.076	4.031	0.848	0.479 + 2200.5
LSTM Original Features	0.0143	0.004	0.193	0.988	0.0188	0.0062	0.235	0.9912	120.58

Table 3.5: Comparison among best models for each feature transformation

associated with a drawback in the precision of predictions made in the corresponding smaller dataset. In order to achieve a better trade off between bias and variance, more instances are preferred. Hence, the tree ensemble learner outperforms the rest of the regression models,including LSTM. However,even for the most optimal model using this feature transformation the adjusted R^2 value does not exceed 85% , which is really low compared to the overall results from the previous experiment.

Both feature transformations: convolution and transient-wise approach require additional time for data preparation. This leads to optionally using the original features measured by PDG data and flow meters to build a predictive model. The absence of a significant relationship between this set of features and the target, is shown by a really low performance of the simple regression models that we tested. Deep learning ,otherwise, doesn't require much feature handcrafting but it can learn from raw data as well. Hence, LSTM shows the most promising results with an approximately 99% determination coefficient for both train and test set and a considerably low error rate, using the originally measured features.

3.2.3 Model comparison

The comparison of the most optimal models for each feature transformation is done in terms of :

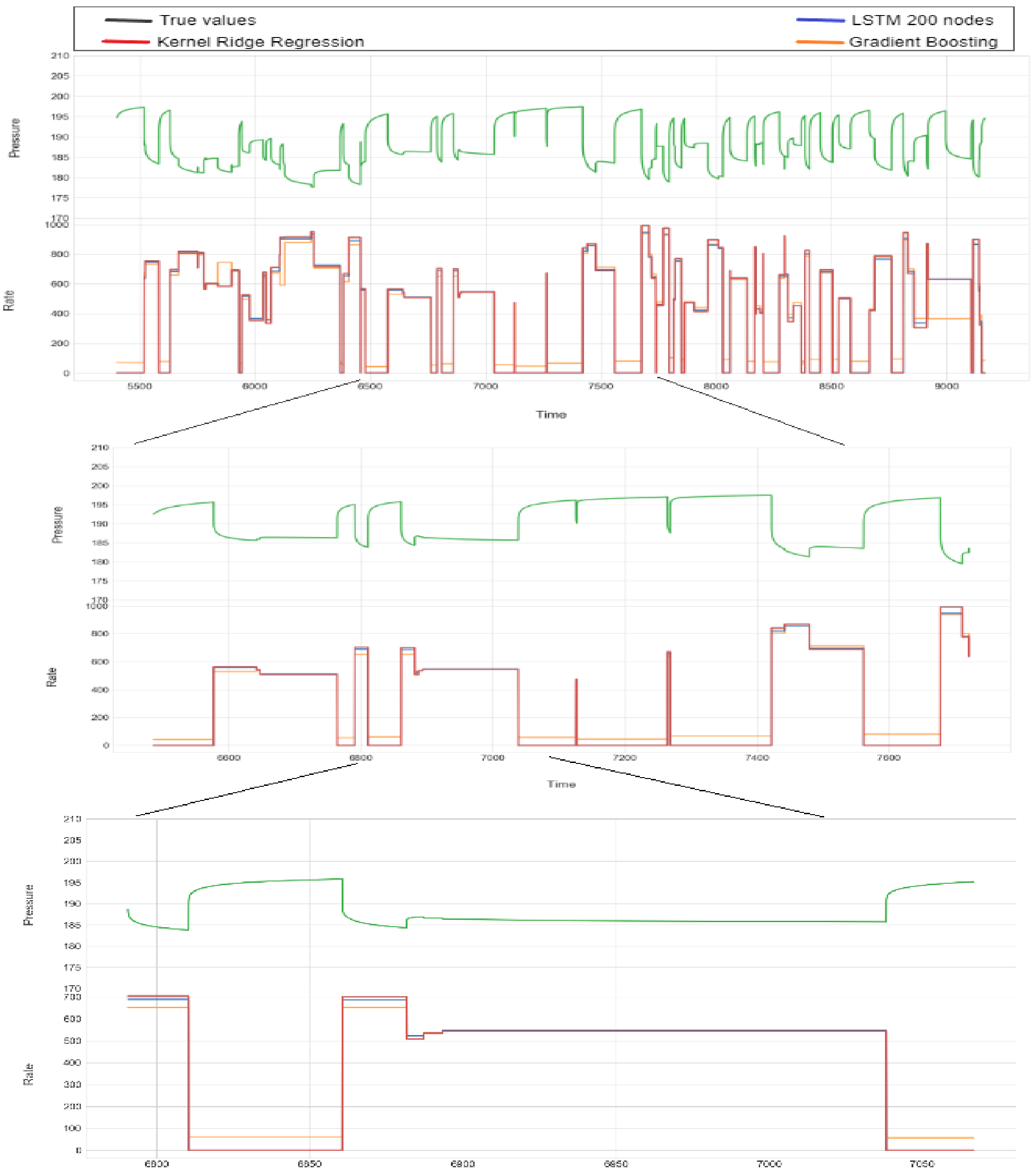


Figure 3.5: Prediction graph of the best models for each feature transformation

1. Accuracy: Gradient Boosting applied in the dataset transformed by the transient-wise approach shows by far the highest error rate, which can also be easily noticed in the prediction graph 3.5 . The most accurate happens to be Kernel Ridge Regression with the convolution feature transformation showing an exceptionally low error rate.

2. Time complexity: Considering only the model fitting time, Gradient Boosting is the fastest algorithm, followed by Kernel Ridge and leaving LSTM to be the most time demanding. However in a bigger picture,adding up the data preparation time

for the convolution approach, Kernel Ridge results in a higher time complexity than LSTM.

With regard to their needs,users can choose the most suitable and optimal model to reconstruct the missing flow rate values. In case the aim is a closer learn of the flow rate pattern and computational expenses are a worth-taken risk ,then Kernel Ridge Regression & Convolution Feature Transformation is the right data-driven solution. If our focus is not in data engineering, but in a powerful model who can learn fast without explicitly crafting the data then LSTM & Original Data is an ideal choice.

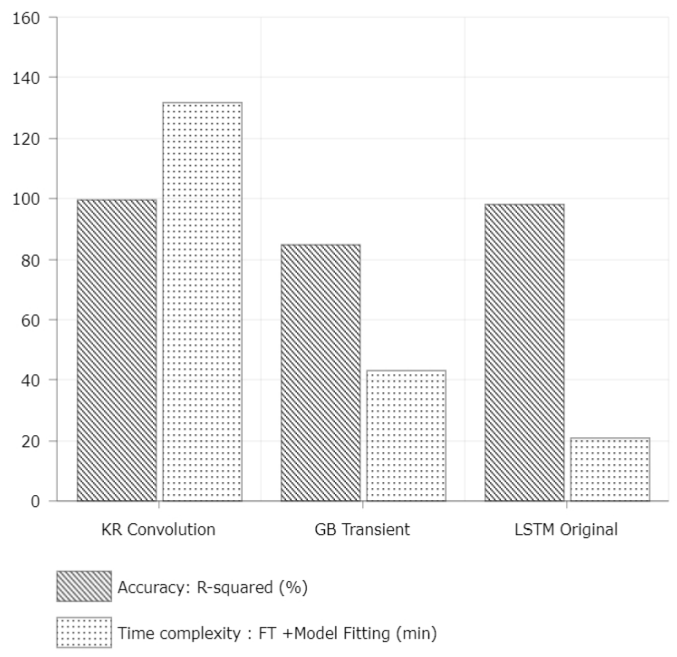


Figure 3.6: Performance comparison

For further work we are using Kernel Ridge Regression as the best model, since we are more interested in the high accuracy this approach demonstrates.

3.2.4 Model application

Subsequent to understanding the well data and finding an algorithm to learn the pattern,our goal is to reconstruct the missing flow rate values.

Until now we tested the case when missing values occurred in the end of the time interval, but in a real-life situation the absence of the flow rate values will be spread irregularly over time. Several flow rate points are dropped in our dataset, to test the performance of Kernel Ridge Regression in predicting these missing values after being trained on the present data. As we can see from the first graph in figure 3.7, the model demonstrates a

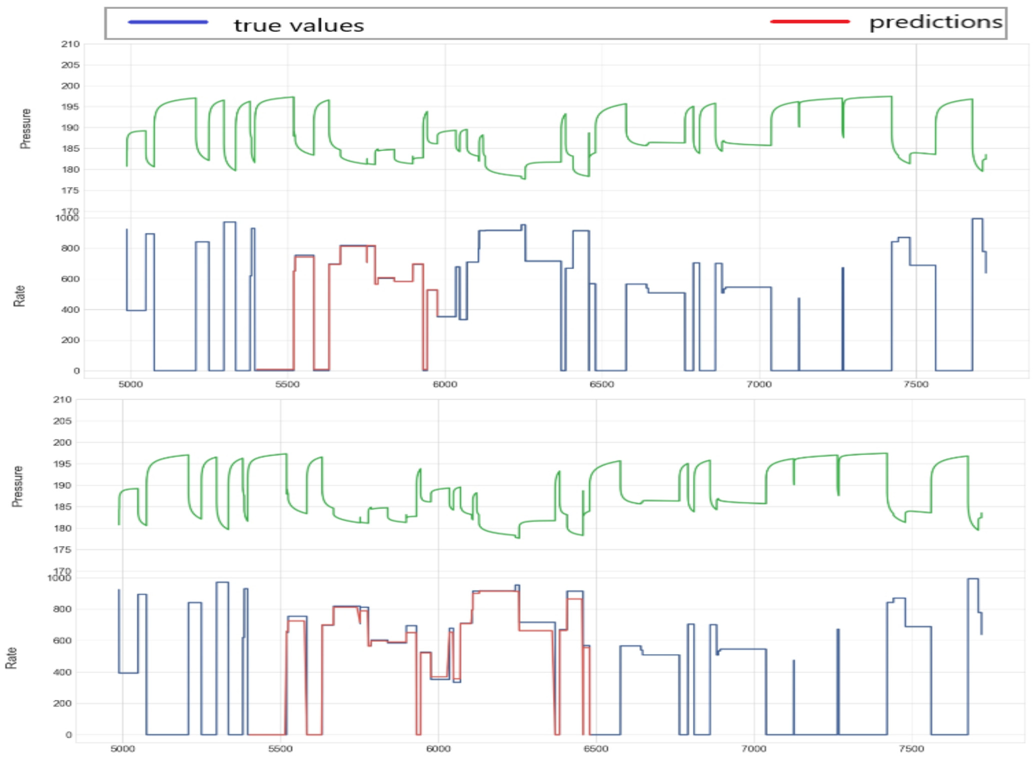


Figure 3.7: Using Kernel Ridge Regression to reconstruct missing flow rate values with different splitting criterion

good fitting as it fills the gaps in flow rate data closely to its true values and it shows a smooth transition from true values to predictions.

However, the proportion of flow rate missing data is sometimes higher than demonstrated in the first figure, not following the ideal split where training set covers most of the dataset. The splitting criteria is to get enough information for training the model and explicitly more data means more accurate model. Hence,this explains the deterioration of the machine learning model when the proportion of missing values in the well data increases. Therefore we should pay attention to this issue before applying the predictive model to our collection of well data.

Chapter 4

Machine Learning model deployment

In the paper published by Sculley et al. in 2015 [19] "Hidden Technical Debt in Machine Learning Systems", they highlight that in real-world Machine Learning (ML) systems, only a small fraction is comprised of actual ML code. There is a vast array of surrounding infrastructure and processes to support their evolution. They also discuss the many sources of technical debt that can accumulate in such systems, some of which are related to data dependencies, model complexity, reproducibility, testing, monitoring, and dealing with changes in the external world.

The process of releasing ML website into production is reliable and reproducible, leveraging automation as much as possible. In this chapter we will explore how we can prepare a machine learning model for production and deploy it inside of a Python Web application. Using the most optimal algorithm from our experiments, we train a prediction model using the labeled input data, integrate that model into a simple web application, which is then deployed to a production environment.[20] Figure 4.1 shows the high-level process.

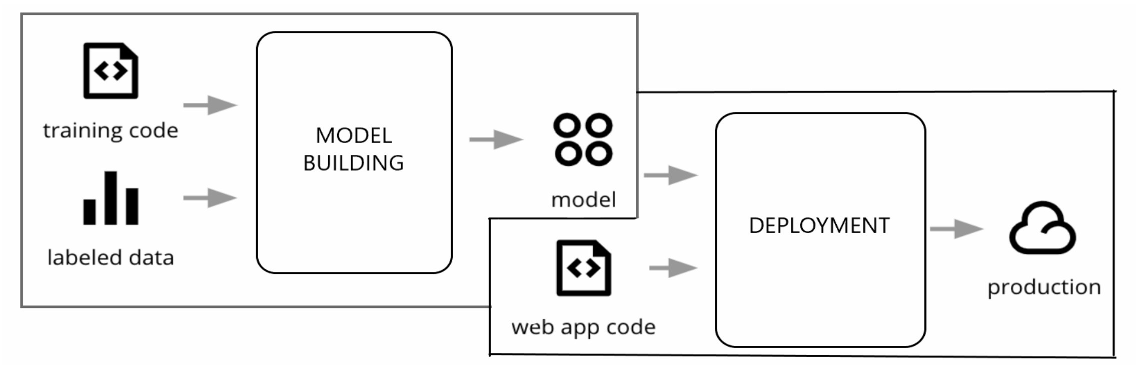


Figure 4.1: [20] Initial process to train our ML model, integrate it with a web application, and deploy into production

4.1 Requirement specifications

The process starts by determining the requirements that the web application will fulfill in terms of performance and complexity. Among different ML system architectures, we will use Rest API ,hence the model will be trained locally and the predict function will be exposed via flask where we can upload our test batch for prediction.

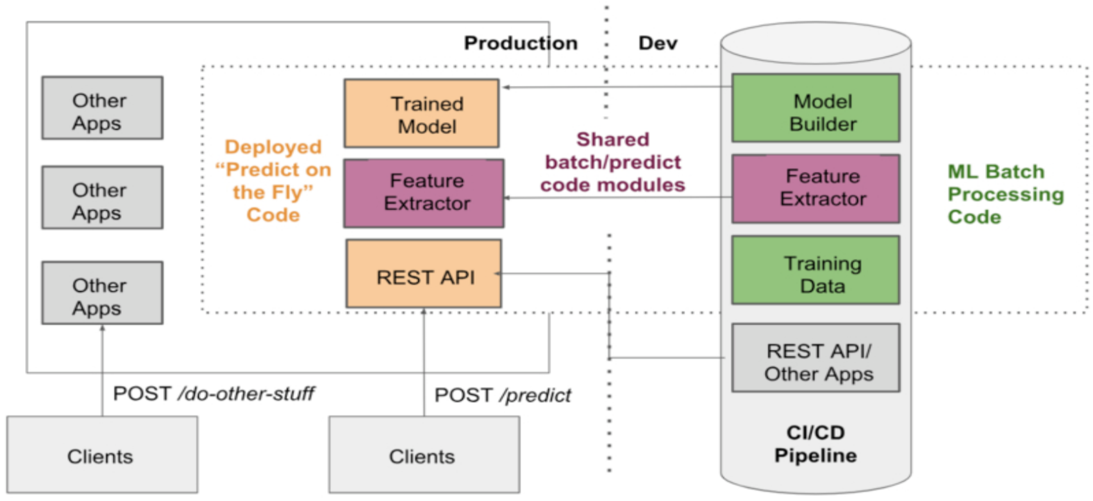


Figure 4.2: [21] Initial process to train our ML model, integrate it with a web application, and deploy into production

As there are many options for creating web applications, the following choices were made:

- In order that the language used in the research environment matches the production environment, the Flask web framework was chosen which is written in Python. This will bring ease in development because of the wide availability of libraries in data preprocessing.
- Tight iOS integration was implemented with HTML along with various CSS and JavaScript tweaks to polish the front end. In addition the Bootstrap framework was chosen for its capabilities in building responsive user interfaces.

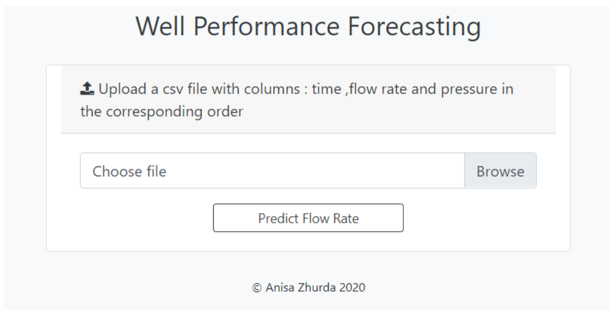
4.2 Flask web application

With regards to our goal to take full advantage of our machine learning model , we are following these steps to build the web application :

1. It is advisable to start by creating a new virtual environment for this project, to avoid any dependency issues.
2. The most optimal machine learning model will be saved in a pickle file. Pickling is a way to convert a python object into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script (app.py).
3. A flask environment will be built which will wrap the trained machine learning model into an API endpoint and enable it to receive features from a csv file through GET requests over HTTP/HTTPS and then return the predictions after de-serializing the earlier serialized model.
4. The input dataset will go through a pre-processing pipeline, to make the set of features in the right format to be fed to the model.
5. The flask script will be uploaded along with the trained machine learning model , and then we make requests to the hosted flask script through a website.
6. To test the prediction endpoint, we can use Postman. Postman is a collaboration platform for API development. Postman's features simplify each step of building an API and streamline collaboration so you can create better APIs — faster [22].

An overview of the website is given with the following main functionalities:

- Upload the dataset as a csv file with the specific order of columns :time,flow rate,pressure.

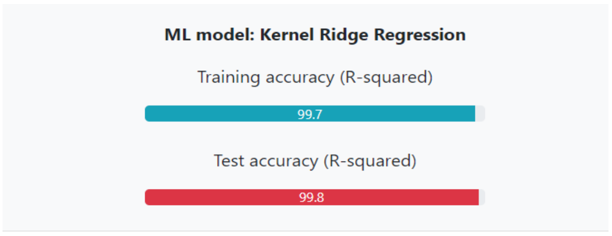

Well Performance Forecasting

Upload a csv file with columns : time ,flow rate and pressure in the corresponding order

© Anisa Zhurda 2020

Figure 4.3: Website: Step 1

- Model performance expressed via percentage R-squared is shown for both train and test set.



ML model: Kernel Ridge Regression

Training accuracy (R-squared)

99.7

Test accuracy (R-squared)

99.8

Figure 4.4: Website: Step 2

- A prediction graph with the reconstructed flow rate values is shown in the following section.

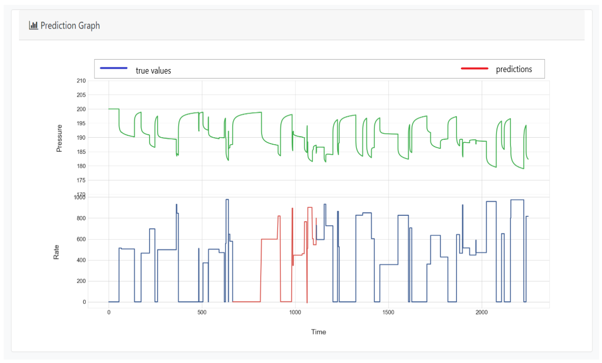
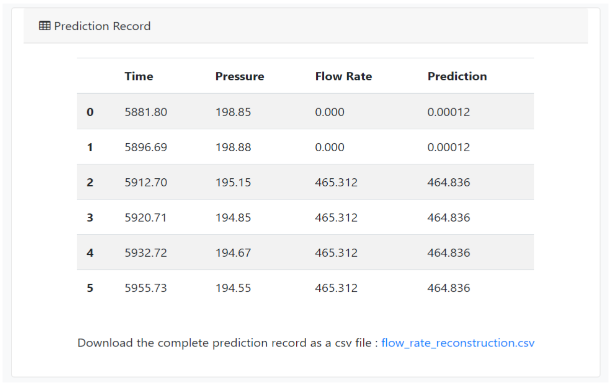


Figure 4.5: Website: Step 3

- In order to take a glimpse of the prediction values, a table of the new dataset is shown with both actual values and predictions. This dataset can be downloaded as a csv file for further usage.



Prediction Record

	Time	Pressure	Flow Rate	Prediction
0	5881.80	198.85	0.000	0.00012
1	5896.69	198.88	0.000	0.00012
2	5912.70	195.15	465.312	464.836
3	5920.71	194.85	465.312	464.836
4	5932.72	194.67	465.312	464.836
5	5955.73	194.55	465.312	464.836

Download the complete prediction record as a csv file : flow_rate_reconstruction.csv

Figure 4.6: Website: Step 4

Chapter 5

Conclusion and Future Work

In this study, several Machine Learning models and one Deep Learning model were applied to a synthetic single-well dataset modeled in various forms, with the aim of an accurate and time-effective reconstruction of missing flow rate values. The main conclusions derived by this work are as following :

- First, we acknowledge the importance that feature transformation has on the performance of a predictive model. The more significance we find between features and the target, the better results are achieved. Modeling the features with convolutional method increased the prediction accuracy in simple regression models, and showed a significant decrease of error rate for Kernel Ridge Regression which captures the well behaviour in more details.
- On the other hand,it was shown that data modeling can be really time consuming, shifting the attention to Deep Learning. The experiment demonstrated how flexible LSTM is in learning patterns from complex data that are not handcrafted and show a non-linear relationship. Even though LSTM has the most demanding model-fitting time, the total prediction time including the data preparation is reasonable and it is justified by the low error-rate.
- Releasing the Machine Learning model into production is as important as building the model. A micro-framework written in Python, known as Flask, enables the use of the Machine Learning model by other end-users in a less challenging way. However, managing the life-cycle of the ML web application is more complex due to overtime degradation.

- Exploring the impact of machine Learning in oil and gas industry is a never-ending process, which by far has shown really promising results with a significant ability to generalize to new and unseen situations. "Data is the new oil" is a metaphoric quote first used in 2006 by Humble [23]. Since then it is continuously proven that proper data handling is the key of profitable decision-makings. However it's important to highlight the limitations of these machine learning applications according to the requirements in the oil and gas industry. A high proportion of missing values can result in deterioration of the predictive model ,hence having some prior knowledge of the well data is critical in the performance of machine learning.

Possible future directions can be addresed according to testing the flexibility of machine learning in real-field datasets corrupted with noise or in a more diverse dataset with additional features that require different feature-engineering approaches. Furthermore,the deep learning usage in oil and gas is just the tip of the iceberg as far as what complexity it has to offer and what parameter optimization can be applied. Finally, the web application is just a demonstration to utilize the machine learning model for well data, but it can be improved with more functionalities and better user interface.

List of Figures

List of Tables

Appendix A

Appendix

Feature transformation: Convolution method

```
def transform_conv(data_1):

    feature1=[]
    feature2=[]
    feature3=[]


    for i in range(1,len(data_1)):
        print(i,i/len(data_1))
        if i%10==0:
            clear_output()
        f1=0
        f2=0
        f3=0
        for j in range(1,i+1):
            f1=f1+(data_1[2][j]-data_1[2][j-1])
            f2=f2+((data_1[2][j]-data_1[2][j-1])*(math.log(data_1[0][i]-data_1[0][j-1])))
            f3=f3+((data_1[2][j]-data_1[2][j-1])*(data_1[0][i]-data_1[0][j-1]))

        feature1.append(f1)
        feature2.append(f2)
        feature3.append(f3)

    data_conv=pd.Dataframe()

    data_conv["feature_1"]=feature1
    data_conv["feature_2"]=feature2
    data_conv["feature_3"]=feature3
    data_conv["flow_rate"]=data_1[1:len(data_1)][1]

    return data_conv
```

Feature transformation: Transient-wise approach

```
def transform_transient(data_1):
    uniq_val={}
    for i in data_1[1].unique():
        data_temp=data_1[data_1[1] == i]
        uniq_val[i]=data_temp.index.values

    for key,item in uniq_val.items():
        transient_time=[]
        for k,g in groupby(enumerate(item),lambda x:x[0]-x[1]):
            index=list(map(itemgetter(1), g))
            transient_time.append(data_1.iloc[index][0].values)
        uniq_val[key]=transient_time

    rate_const=[]
    start_time=[]
    pressure=[]
    duration=[]
    for key,items in uniq_val.items():

        for i in range(len(items)):
            min_time=min(items[i])
            max_time=max(items[i])
            delta_time=max_time-min_time

            min_pressure=data[data[0]==min_time][2].values[0]
            max_pressure=data[data[0]==max_time][2].values[0]
            delta_pressure=max_pressure-min_pressure

            duration.append(delta_time)
            start_time.append(min_time)
            rate_const.append(key)
            pressure.append(delta_pressure)

    df_transient = pd.DataFrame()

    df_transient["start_time"]=start_time
    df_transient["delta_time"]=duration
    df_transient["delta_pressure"]=pressure
    df_transient["constant_rate"]=rate_const

    return(df_transient)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
clf = LinearRegression()
```

Ridge Regression

```
from sklearn.linear_model import Ridge

params={'alpha': np.logspace(-10, 2, 300),"fit_intercept":[True,False]}
rdg_reg = Ridge()
clf = GridSearchCV(rdg_reg,params,cv=2,verbose = 1, scoring = 'neg_mean_squared_error')
ridge_model = Ridge(**clf.best_params_)
```

Kernel Ridge Regression

```
from sklearn.kernel_ridge import KernelRidge
kernel_rdg = KernelRidge(kernel="rbf")

params={"alpha": np.logspace(-10, 2, 300), "gamma": np.logspace(-10, -1, 100)}

clf = GridSearchCV(kernel_rdg,params,cv=2,verbose = 1, scoring = 'neg_mean_squared_error')

kernel_ridge_model = KernelRidge(**clf.best_params_)
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor

clf = GradientBoostingRegressor(n_estimators=1000, learning_rate=0.5,
    max_depth=2, random_state=0, loss='ls')
```

LSTM

```
for lstmsize in range(20,200,10):
    lstmdropout=0.4
    generaldropout=0.5
    reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='loss',
                        factor=0.5, patience=5, min_lr=0.001)

    m = Sequential()
    m.add(LSTM(lstmsize, input_shape=(n ,19), recurrent_dropout=lstmdropout))
    m.add(Dropout(generaldropout))
    m.add(Dense(n,activation='relu',kernel_regularizer=regularizers.l2(0.001)))

    m.compile(loss='mse', optimizer='adam',metrics=['accuracy'])

    history=m.fit(train_X,train_Y, epochs=1000,
                batch_size=10, verbose=0, callbacks=[NEpochsLogger(50),reduce_lr])
```

Flask application instance

```
app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def home():
        return render_template('index.html')


@app.route('/predict',methods=['POST'])
def predict():
    data = pd.read_csv(request.files.get('data_file'))

    data_1=transform_conv(data)

    data_1 = data_1.astype('float32')
    scaler = MinMaxScaler(feature_range=(0, 1))
    dataset = scaler.fit_transform(data_1)

    TESTx=dataset[:,:-1]
    TESTy=dataset[:,-1]

    prediction=model.predict(TESTx)
    score=model.score(TESTx,TESTy)
    prediction=prediction.reshape((prediction.shape[0],1))
    prediction_results = concatenate((TESTx, prediction ), axis=1)
    prediction_scaled = scaler.inverse_transform(prediction_results)

    plt.plot(prediction_scaled[0],prediction_scaled[1])

    figfile=io.BytesIO()
    plt.savefig(figfile,format="png")
    figfile.seek(0)

    figdata_png=base64.b64encode(figfile.getvalue()).decode("ascii")
    prediction_scaled=prediction_scaled.head(n=10)
    data.columns=["time","rate","pressure"]

    return render_template('index.html',prediction_text=round(score,4)*100,
        tables=[prediction_results.to_html(classes='data')],
        titles=["time","rate","pressure"],results=figdata_png)

if __name__=='__main__':
        app.run(debug=True)
```

Bibliography

[1] Exploring machine learning developments in the oil and gas industry. *Project of Harvard Businnes School*, November 2018.

[2] Vinodkumar Raghothamarao. Machine learning and ai industry shaping the oil and gas industry. *Pipeline,oil and gas news*, 2020.

[3] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, January 2000. ISSN 0018-8646. doi: 10.1147/rd.441.0206. Conference Name: IBM Journal of Research and Development.

[4] Dan Hebert and Alex Misiti. The Growing Role of Artificial Intelligence in Oil and Gas. *Engineering360*, 2016. doi: https://insights.globalspec.com/article/2772/the-growing-role-of-artificial-intelligence-in-oil-and-gas.

[5] Chuan Tian and Roland N. Horne. Recurrent neural networks for permanent downhole gauge data analysis. *SPE Annual Technical Conference and Exhibition, 9-11 October, San Antonio, Texas, USA*, . doi: https://doi.org/10.2118/187181-MS.

[6] Yang Liu and Roland N. Horne. Interpreting pressure and flow-rate data from permanent downhole gauges by use of data-mining approaches. *SPE Journal 18(1)*, pages 69–82, 2012.

[7] Yang Liu and Roland N. Horne. Interpreting pressure and flow rate data from permanent downhole gauges using convolution-kernel-based data mining approaches. *SPE Western Regional AAPG Pacific Section Meeting 2013 Joint Technical Conference, Monterey, California*, 2013a.

[8] Yang Liu and Roland N. Horne. Interpreting pressure and flow rate data from permanent downhole gauges using convolution-kernel-based data mining approaches. *SPE Annual Technical Conference and Exhibition, New Orleans, Louisiana*, 2013b.

[9] Chuan Tian and Roland N. Horne. Applying Machine Learning Techniques to Interpret Flow Rate, Pressure and Temperature Data From Permanent Downhole Gauges. April 2015. doi: 10.2118/174034-MS.

[10] Chuan Tian and Roland N. Horne. Machine Learning Applied to Multiwell Test Analysis and Flow Rate Reconstruction. . ISBN 978-1-61399-376-7.

[11] A. Aggarwal and S. Agarwal. ANN Powered Virtual Well Testing. January 2014. doi: 10.4043/24981-MS.

[12] M. Korjani, Andrei Popa, Eli Grijalva, Steve Cassidy, and I. Ershaghi. A New Approach to Reservoir Characterization Using Deep Learning Neural Networks. January 2016. doi: 10.2118/180359-MS.

[13] Cristina Heghedus, Anton Shchipanov, and Chunming Rong. Advancing Deep Learning to Improve Upstream Petroleum Monitoring. *IEEE Access*, 7:106248–106259, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2931990. Conference Name: IEEE Access.

[14] Implementing Gradient Boosting Regression in Python | Paperspace Blog.

[15] George Loukas, Tuan Vuong, Ryan Heartfield, Georgia Sakellari, Yongpil Yoon, and Diane Gan. Cloud-based cyber-physical intrusion detection for vehicles using Deep Learning. *IEEE Access*, 6:3491–3508, December 2017. doi: 10.1109/ACCESS.2017.2782159.

[16] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[17] Jason Brownlee. A gentle introduction to scikit-learn: A python machine learning library. *Machine Learning Mastery*, 2014.

[18] François Chollet et al. Keras. https://keras.io, 2015.

[19] Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, and Dan Dennison. Hidden technical debt in machine learning systems. *NIPS*, pages 2494–2502, 01 2015.

[20] Christoph Windheuser Danilo Sato, Arif Wider. Automating the end-to-end lifecycle of machine learning applications. September 2019.

[21] Bamigbade Opeyemi. Deployment of machine learning model demystified. *Towards Data Science*, November 2019.

[22] Api platform: Api tools and solutions from postman. https://postman.com.

[23] Data is the new oil. *ANA Senior marketer's summit, Kellogg School*, 2006.