mlflowrate: A Machine Learning Package for Predicting Fluid Rates in Offshore Wells

Kevin Fung
kkf18@imperial.ac.uk
Github Alias: kkf18

Imperial College London
Department of Earth Science and Engineering
MSC Applied Compuational Science and Engineering: ACSE 9

Internal Supervisors:
Prof. Olivier Dubrule: o.dubrule@imperial.ac.uk
MSc Lukas Mosser
lukas.mosser15@imperialc.ac.uk

External Supervisors:
Dr. Meindert Dillan
meindert.dillen@wintershalldea.com
Peter Kronberger
peter.kronberger@wintershalldea.com

The following thesis project was carried out in collaboration with Wintershall Dea

Abstract

Machine learning is becoming a prominent tool in predicting the fluid rates of wells. To leverage the power of machine learning, it is crucial for well data to be properly cleaned and sorted for models to produce accurate predictions. mlflowrate is a data integration and machine learning package which provides analysts the ability to quickly produce flow rate predictions from raw well data files. The developed software is tested on well data provided by Wintershall Dea. An analysis into the data is conducted with the software, and finally liquid flow rates are predicted for the specific well. It was found that using random forest regression on a small feature set produced the best R2 model score of 0.881, which shows a promising indication to apply such modelling techniques to other oil wells.

Keywords: Data Integration, Production Rates, Machine Learning

1. Introduction and Significance

Accurate measurement of multiphase flow rates is a critical aspect in enhancing production efficiencies in wells, particularly in offshore oil wells. Monitoring these characteristics enable better informed maintenance decisions for operators. One instance is scale detection, where liquid rate can dangerously drop due to rapid scale formation caused by chemical imbalances [1], however, not all significant drops are caused by scaling and this generates uncertainty. Therefore, by using accurate flow rate estimations uncertainty about root causes can be reduced.

1.1. Literature Review

Traditionally, continual multiphase flow rate measurements of individual wells are not available. Sin-

gle well production flows are instead commonly diverted to a test separator for independent phase flow measurements. Frequency of these test separator measurements are typically monthly, but operator-dependent. Due to the small number of these measurements, operators have two options for estimating finer resolution flow rates. The first relies on multiphase flow meters (MPFM) for direct flow measurement in individual wells. The second involves virtual flow meters (VFM) that are mathematical models of flow based on the available measured data. It is common in industry to opt for VFMs due to its cost efficiency and lack of maintenance [2].

VFMs can be labeled as either data-driven or hydrodynamical. Hydrodynamical models are constructed on classic fluid mechanics to estimate flow rates. Amin [3] investigated the performances of commercially available Pressure-Volume-Temperature based VFMs. It was concluded that these models produced robust flow estimations especially for low water cut flows, however, noticeable drawbacks included their sensitivity to anomalous inputs, and poor estimation of gas oil ratios.

In contrast, data-driven models rely solely on the available data with no a priori information to derive a model for estimating flow rates. Models developed this way tend to have an underlying structure difficult for interpretation, yet they are more general for applications. Moreover, they have the potential to infer unknown relationships in data whereas hydrodynamical VFMs are limited by their explicit modelling.

Presently, machine learning based VFMs have become the focus of attention in the field. Omrani et al [4], investigated the feasibility of neural networks (NN) to estimate flow rates with simulated and field datasets. A sufficiently sampled field dataset allowed them to produce a Long-Short Term Memory (LSTM) model to estimate daily liquid rates with relative percentage errors 25% within its mean error of 1%. However, their case to using synthesised datasets to alleviate unavailability of flow rate data was not sufficient. This is because their problem became a question of fitting a neural network to the generative model (which synthesised the datasets) rather than fitting to the underlying distribution encountered in reality.

Adrianov [5], also used LSTMs to forecast liquid flow rates yielding positive results, capturing the trend and demonstrating the model robustness to artificially induced noisy inputs. Ristanto [6], explored the suitability of ten popular machine learning models to calculate production flow characteristics including rates. It was concluded that linear regression, support vector machines (SVM), and fully-connected neural networks produced accurate predictions. Hasanvand [2], also used fully-connected neural networks to predict flows accurately, however, it was highlighted that for consistent predictions regular learning on new data for the model was essential.

1.2. Project Proposal

There is a positive indication from literature that machine learning can be an effective option for accurately predicting fluid rates. However, the application for such machine learning approaches can prove to be cumbersome and time consuming. In addition, within industry, different qualities and quantities of data are often provided to the analyst. Thus, in predicting flow rates on an industrial level, there is no single machine learning approach that will be applicable for all data situations.

In regards to the prominent problem and the continual rise of machine learning to predict flow rates, the proposed thesis aims to develop a high-level software package to predict liquid flow rates with a data integration system for commonly encountered oil well data. The software is intended for the Apache Spark data framework within an Azure Databricks environment. The machine learning library Scikit Learn, will be the main dependency for machine learning algorithms. A use case of the software with data provided by Wintershall Dea shall be presented along with an analysis into the data and predictions itself.

The software is built with Python because of it's substantial use in data science applications. An incremental software development process is followed which ensured functional validity and consistent code verification throughout the project. In addition, an object-oriented code structure for feature extensibility was implemented.

The data used in this project is known to be unclean and disorganised. This occurs for several reasons. For example, the installation of a sensor for a new measurement caused the logged measurements to become inconsistent with other logged measure-

ments that have been in operation for longer. Missing and erroneous characters may also appear in the data due to the outputting formats of different data logging software. In regards to the machine learning problem, there were relatively few labelled liquid flow rate samples amongst a large quantity of unlabelled samples in the data.

After an initial incremental build is developed, the provided data was used for validating the functionality of the software. This further allowed an investigation into the data to be performed concurrent to the software development. Two essential model error estimation methods which tackle the labelled sample insufficiency are implemented for model comparisons. The machine learning approaches implemented were regression and non regression models, and it was found that data cleaning and feature engineering played a vital role in improving the accuracy of model predictions.

1.3. Software Requirements

The software has been intentionally designed to handle Spark data structures from Apache Spark, a popular cluster computing framework for processing big data. Frameworks like Apache Spark currently provide low-level programming usage for data processing, and there is little to no support for a systematic workflow in preparing datasets and exploring big data. Therefore, analysts are required to have an extensive experience in using frameworks like Apache Spark and SQL. Furthermore, when working on data processing in teams, using these frameworks can reduce code interpretability given the declarative programming style that is integrated within these frameworks. Alleviating the need for such experience in Apache Spark and improving code interpretability therefore falls within achieving our objective. The following therefore lists the functionalities which the software aims to fulfill:

1. Integrate commonly encountered tabulated well data into a suitable format for machine learning.
2. Clean data with anomalies and duplicates.
3. Construct datasets from multiple sources of data.
4. Allow users to predict liquid flow rates with different machine learning models.
5. Provide suitable error performance methods for insufficiently labelled data.
6. Visualise data and predictions.

2. Software Development Life Cycle

In developing the software, an incremental life cycle model was adopted because it allowed for concurrent model explorations whilst keeping to project time constraints. The advantage was that upon each incremental build, the software modules can be validated through a phase of predictive flow rate exploration. Any functional issues or code bugs bypassing unit tests would then be identified during this phase, and finally a new increment of software development would be carried out again. For version controlling of software, Github was used.

3. Technical Back-end

The software architecture is fundamentally structured on a nested class with three instantiated sub classes to facilitate the data workflow. The outer class, WorkFlow, manages the flow of data by receiving processed data from the sub classes, as well as tracking the sub classes which the user has used. The three sub classes follow the general data workflow for predictive modelling:

1. Data cleaning and sorting
2. Constructing datasets
3. Exploring and modelling data

The classes corresponding to these phases are DataIntegration, AssembleDatasets, and DataExploration. Users are exposed to the aforementioned classes through the nested class structure, however, three back-end classes also exists for the software to function accordingly. A parent class named Base is developed to extend essential data reformatting tools as well as a general visualisation feature to the three inner classes. The other two classes are intended to be object storage for data. The Dsets class is designed to hold the components of a constructed dataset that is typically used in machine learning models. Whereas, the Results class provides an adaptive storage object for different evaluation metrics of a given prediction

model. The software architecture and intended functional workflow is illustrated by the architectural design diagram in Figure 1.

To summarise the software file structure, the four front-end classes are separated into individual python modules. These are workflow.py, data.py, datasets.py, and dataexplore.py. The Dsets class resides in the datasets.py module for its usage in the dataset phase, and the Results class is put in the dataexplore.py module for similar reasons. The class Base is contained in an independent module base.py.

4. Design Rationale and Code Implementation

The following section explains the design and code implementation of the key features across the overarching functionalities of the software, that is the flow of data management, data integration, dataset assembly, and data modelling.

4.1. Workflow Management

From an end-user perspective, the choice of a nested class structure provides the ability to instantiate multiple flow objects containing different sets of pre-processed data and exploratory results. Furthermore, code readability is significantly improved from the structure of calling nested class methods. For example, instantiating and naming a workflow object, regressionmodels, then next calling the method regressionmodels.datasets.make_set() indicates to users the code is assembling a dataset for an exploration into regression modelling.

The key feature to manage the flow of manipulated data on to subsequent classes is achieved via the next_phase() method. This is where a collection of nested if statements instantiate the next functional sub class when called. A benefit to this is so that users are limited to the methods in the current class in the data workflow, and thus it prevents the misuse of any features further along the workflow. This may also be considered as a method to enforce the data workflow upon users.

4.2. Data Integration

The next main functionality is to enable users to sort and clean multiple sources of data in preparation for constructing datasets.

Upon this stage, the analyst typically has two types of formatted well characteristic data to integrate into the workflow. The first type of formats are excel datasheets, whereas the second type of formats are csv files containing a time column, tag-name column, and data column. The major issue with the latter data type is that the tag-name typically comprises of two pieces of metadata relating to its measurement. These are often the origins of the measurement, and the type of measurement collected. Therefore, the provided data must be separated and sorted into its respective origins and its measurement type. Furthermore, it is also common in industry for these tag-names to be word coded, and thus requires a code dictionary to understand what each tag-name represents.

In regards to this, the DataIntegration class contains two styles of organising DataFrames to enable the sorting of asymmetric data that can arise from sorting the csv files. The first style of DataFrame organisation is a ”grouped dictionary” organisation. Where a set of individual DataFrames containing a time column and measurement column are collected into a dictionary, and whose key names correspond to the measurements that they contain. The intuition behind this style of organisation is to separate out incontiguous sets of data where some features may not contain the same quantity of samples as others. The second style of organisation is the use of the DataFrame itself, whereby it enforces data to become symmetric. However, in the presence of incontiguous data, null values will be generated as a feature of the Spark DataFrame. In essence, the combination of the two styles enables the integration of disorganised data into a suitable dataset for machine learning, and certain cleaning features are significantly optimised by utilising one style over the other.

4.2.1. Data Reorganisation

The data reorganisation method, organise_data(), is the key feature that integrates the unsorted csv and excel data into the two organisational styles for DataFrames. For the end-user, three options are provided for the type of reorganisation desired to the target data. The first option, ”date_tag_val_col”, addresses the typical csv data formats for oil well data and collects the distinct tag-names into the individual DataFrames within

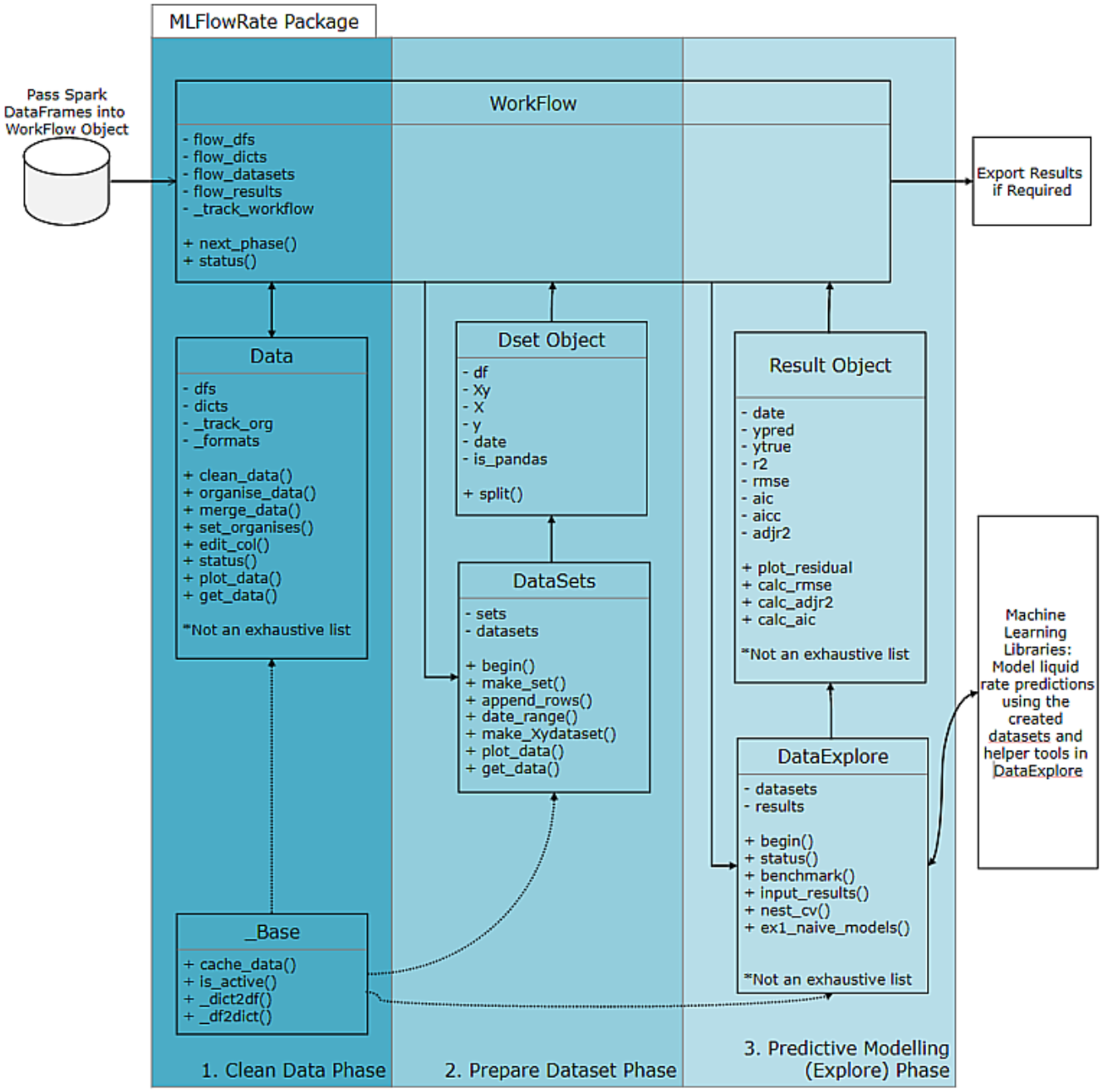

Figure 1: UML styled Architectural Design Flow of the MLFlowRate package. Dashed lines indicate inheritance relationships.

the grouped dictionary style. The corresponding DataFrame is also produced at this stage to ensure the data maintains the two organisational types throughout the data cleaning process. However, as mentioned previously, null values will be generated if the distinct tag-names held inconsistent samples to one another. In consideration that the tag-name holds further information for sorting, two more extensible features, "distinct_oilwells" and "change_sensor_names", passed in as keyword arguments enable the creation of multiple grouped dictionaries corresponding to the distinct well origins from the set of sensor measurements, as well as the decoding of tag-names. The keyword arguments require a dictionary with key value pairs corresponding to the translation of tag-names. In addition, the translations are required to be written in the regular expressions (re) style. Both extensions rely on the re library for finer control of selecting specific groups of characters in the tag-name.

The second option, "mult_col", is designed to generate the grouped dictionary style for the excel data since the data is already consistent. In contrast to this, the final option, "dict_col", generates a DataFrame from a grouped dictionary instead. The algorithms for generating the grouped dictionary and symmetric DataFrames are implemented within the Base class and is called within the organise_method(). These two algorithms can be found within the _dict2df and _df2dict methods. The former holds more importance due to the possible inconsistency in data, wheras the latter simply involves an iteration of optimised Spark DataFrame methods. To describe the _dict2df algorithm, the process begins with collecting the first key-value pair in the dictionary. Next it iteratively left-joins the subsequent key-value pairs back to a Spark DataFrame. However, this method is not suitable for inconsistent data, since the first DataFrame in the dictionary may be significantly larger than the others and thus null values would be generated. Passing in the option, "inconsistent", forces the algorithm to initially select the smallest sampled data feature in the grouped dictionary instead, and therefore eliminates the occurrence of null values being generated due to size inconsistency. The drawback to using this method however, is that it comes with a computational cost. Due to the distributed data architecture of Spark DataFrames, querying the number of samples of a DataFrame is time consuming, especially for larger DataFrames. Therefore, making iterative queries into the grouped dictionary DataFrames can become an increasingly expensive method to run.

To alleviate this issue, a key feature in the software to optimise such queries is implemented as the cache_data() method. Fortunately, Spark includes a unique method, cache(), to be able to retrieve and collect up the distributed data of a Spark DataFrame into cluster memory. The advantage of this is so that the computing cluster does not need to repetitively perform any additional background tasks to search and collect the distributed data for querying. The wrapper method, cache_data() resides in the class so that end-users are able to utilise this functionality across all stages of the workflow.

4.2.2. Data Cleaning

The features provided for cleaning data are encapsulated within two methods, clean_data() and -edit_column(). Both methods rely on the consistent formatting of data, and is therefore intended to be used after data reorganisation. Within the case of operational well data, there are a multitude of susceptible problems which may occur with the measurement data. This may include broken sensor measurements resulting in zero errors, missing measurement values, and duplicate time samples. For our specific well data, the clean_data() method provides three parameter options for cleaning the entire DataFrame, these are "remove_nulls", "remove_char", and "avg_over". We intentionally not implement the removal of zero values for certain measurements like liquid flow rate would actually contain zero valued measurements (for when a well is shut-in). This issue is resolved during the dataset assembly stage, when the user begins to analyse the cleaned data for feature selection. The "avg_over" method is designed for the removal of duplicate time samples since the provided well data is measured on an hourly basis, and duplicates therefore arise within the hour precision. It is intuitive to average over the data to a daily precision instead of dropping these values as duplicate time samples themselves may contain different measurements to each other.

The edit_column() method is functionally intended for users to manipulate the metatypes of individual

columns in the DataFrame. To elaborate, end-users are able to rename the desired feature columns in data as well as recast the column's variable type. The need for recasting arises when importing the data via Apache Spark. The "infer_schema" option provided in Spark's import methods may mistakenly assign an incorrect variable type to a feature column due to the missing values or incorrect characters in the data.

4.2.3. Data Integration Management

To manage data cleaning and ensure the final cleaned data is returned back to the WorkFlow object, an additional feature which enables users to track and set cleaned data is implemented. The _track_org attribute is a dictionary of data names and booleans, where true indicates that data is cleaned and ready to be returned to the WorkFlow. The status() method, displays this tracked information as well as the number of samples, null values, and sample duplicates to assist users in identifying the appropriate cleaning operations required. After cleaning, users can indicate which data is cleaned with the set_organised() method.

4.3. Dataset Assembly

The next stage of the process is to construct the appropriate datasets for predictive modelling. Feature selection and making datasets appropriate for use with machine learning packages are the key functionalities provided by this class. Therefore, a collection of statistical visualisation methods and data transferring methods are implemented.

4.3.1. Feature Selection

The intended process within assembling datasets is to initially generate Spark DataFrames from the multitude of cleaned data with the make_sets() method. This allows for any data manipulation after selecting the features. However, an issue arises where inconsistent data sizes will generate null values in the DataFrame again. Furthermore, the time precision of data sources may be different to each other. To resolve the two problems, the "align_dates" parameter is passed truncating all time columns of the selected features to the specified time precision. The _dict2df() method with the "inconsistency" option enabled is implemented to avoid any inconsistent data sizes.

Three different statistical visualisation tools are also provided to assist the feature selection process. These are distributions(), pairplot(), and pearsons(). To briefly summarise, the distributions() method creates a distribution plot based on the kernel density estimation for each feature in the dataset. The pairplot() method generates a matrix of pairwise bivariate distributions across all the features provided. Finally, the pearsons() method plots a Pearsons correlation matrix. The aforementioned methods are wrapper methods for functions derived from the Seaborn and Pandas libraries, and so they require the DataFrame to be in a Pandas DataFrame format. Spark provides the functionality to transform Spark DataFrames into the Pandas format, however, the computational time for reformatting can be slow and is proportional to the size of the DataFrame. The previously explained cache_data() method partially alleviates this problem only in the case of repeated transformations of the same Spark DataFrame by repeatedly running the same notebook cell.

Further data selection features are implemented and are designed for a finer refinement of the assembled DataFrames. The append_rows() method enables users to move specific samples from one DataFrame to another based on a condition. In the case of different sets of oil well data, certain samples may become relevant to building a fully featured dataset for machine learning. The method, however, has a slightly reduced user experience as it requires users to pass in a lambda function encapsulating a Spark where() method for selecting samples.

4.3.2. Assembly for Machine Learning Packages

The next main functionality is to transform DataFrames into independent sets that work with popular machine learning libraries.

Instantiated Dset objects are essential for holding and splitting the data appropriately into feature, label, and date sets. The make_dataset() instantiates these objects and calls the internal split method to separate out a passed in DataFrame. For the current investigative case, the option to transform the sets into a Pandas DataFrame is implemented so that we can use the Scikit Learn package.

4.4. Data Exploration

Finally, the predictive modelling of liquid rates is encapsulated by the dataexploration class.

Users at this stage are provided the flexibility to work outside the workflow given the multitude of available machine learning libraries. The get_datasets() method returns the created datasets by the software, and after modelling users are able to feed results back into the software for storage and analysis.

A crucial feature to highlight are the general error estimation techniques implemented for the case of a limited availability of labelled liquid rate data, in our use case, less than 150 labelled samples were available. Because of this, adopting the typical approach of splitting into train, validation, and test sets with the labelled data is not as appropriate. We then face the issue of potentially reserving too much training data that is at risk of producing an unreliable general performance estimation.

The first error estimation technique implements the nested k-fold cross validation algorithm for our models. As its name suggests, the nested k-fold cross validation is the nesting of an inner k-fold cross validation loop within an outer k-fold cross validation loop. The advantage of using such an technique is so that an unbiased general error estimate may be found by including the an unbiased hyperparameter tuning of models within the general error estimate [7] [8]. For every inner k-fold loop a set of optimal parameters on the split data is first found. The outer k-fold loop then trains with each optimised parameter sets and averages the outer fold estimates to determine an unbiased generalisation error of the model. The method validate() implements this technique with Scikit Learns cross validation methods. The grid search cross validation technique for inner loop hyper parameter tuning is selected as we can leverage the distributive capabilities of the Azure-Databricks cluster environment. In other words, multiple nested cross validation tests may run simultaneously on individual notebooks each leveraging the clusters computational power.

The second error estimation technique is relatively case specific, in our use case, we are provided a VFM liquid rate model whose relative uncertainty errors lie within a range of 15%. We take advantage of the provided VFM data to assist in determining the general model performance by calculating the root mean squared error (RMSE) away from the VFM's uncertainty boundaries. This provides a partial model error estimation but nonetheless useful. If the model is entirely within the VFM model's uncertainty range, then the model can be considered as a competitor to the VFM. This feature is implemented within the evaluate() method that plots the predictions with the VFM and displays the RMSE from the uncertainty range. The model is intended to be trained on the entire labelled liquid set with the hyperparameters tuned with k-fold cross validation.

Both error estimation methods are designed to be used together for a full analysis into the performance of a model, and therefore reducing the uncertainty in measuring model performances.

The last key feature implemented applies a series of generic machine learning models to the dataset. The fitpredict_naivemodels performs the nested cross validation method to determine the general error estimation and returns the best model predictions trained and tuned on the entire labelled set. The provided models and their parameter tuning ranges are listed below: Furthermore, an extension to the feature is provided, where users are able to fully train and tune every model in the method for a comparative analysis against the VFM data using the evaluate() method. In addition, the feature is designed in line with the software requirement to also predict with a collection of machine learning models.

To provide users a direction in exploratory modelling, four specialised methods have been integrated into the class which encapsulate the four exploratory approaches that have been investigated in this thesis.

4.5. Code Validation and Verification

As explained previously, code validation is thoroughly tested via the incremental development life cycle model. For code sustainability, unit tests for methods especially in the dataorganisation class were integrated in the software development process. Moreover, assertions were important in methods to avoid any user input errors.

5. Code Metadata

The implementation platform for the developed software was in PyCharm, and is intentionally de-

ployed on an Azure Databricks Notebook environment. However, a Jupyter Notebook environment with the Apache Pyspark library preinstalled is also sufficient for deployment. Excluding Apache Pyspark, which is a standard preinstallion in the Azure Databricks Notebook environment, the software dependencies are as follows:

1. numpy
2. matplotlib
3. re
4. pandas
5. seaborn
6. sklearn

In addition, the software version is currently 1.0.0 and access to the software code can be found via the Github repository link: https://github.com/msc-acse/acse-9-independent-research-project-kkf18.git. - Software documentation is described within the Github markdown.

6. Use case: Predicting Liquid Flow Rates

In this section we present the application of mlflowrates to the data provided by Wintershall Dea to predict flow rates. An analysis into the data is first discussed, then an approach to investigating modelled liquid rates is formulated and applied with mlflowrates.

To reinstate our machine learning problem, we have been provided a small set of labeled liquid flow rate data with a large set of unlabeled field measurements for predicting liquid flow rates. A VFM dataset containing liquid flow rate estimations has also been provided and thus is used in estimating model performances.

6.1. Data Overview

Following our data workflow, an initial insight into the data provided must be analysed for the appropriate cleaning and preparation of our datasets. The data provided is from a well located in the North sea, and is known to have maintenance issues predominantly relating to calcium carbonate scaling within the well pipes. Four different sources of time series data were provided for this well. These were the following:

1. **Field data**: Contains accurate sensor measurements recorded on an hourly basis from the well head down to its well bottom.
2. **Test separator data**: Contains measured oil, water, and liquid flow rates, as well as averaged field measurements for VFM modelling.
3. **MIKON estimated flow rates**: Predicted daily flow rates from a VFM currently in operation.
4. **Recorded interventions**: A list of documented man made interferences that occurred over the wells lifetime.

6.1.1. Field Data

Eight types of sensor measurements are collected every hour in the field and spans back to the opening of the well in 2015. These are well head pressure (whp), well head temperature (wht), down hole pressure (dhp), down hole temperature (dht), gas lift rate (glr), gas lift pressure (glp), acoustic sand detection (asd), and finally choke percentage.

The well head is an essential connecting component located at the well surface which directs production flow into the main production line. The well head sensor measurements are installed at this point in the well. Similarly, the down hole measurement sensors, as its description implies, are located at the bottom of the well. The choke, a special valve that controls the production flow rate, is a part of the well head and measurements of choke are recorded as the percentage of choke closure, for example, a hundred percent choke recording indicates a fully closed choke. An acoustic sand detector measures the quantity of sand build up within the choke. The build up of sand is an undesired well interference which may naturally restrict the liquid flow. Finally, the gas lift rate and pressures are measured.

The gas lift is formally known as injecting gas down the well system in order to create a desired pressure difference between the well head and down hole, which increases the production flow rate. Gas lift injection is vital to improving production flow rates, however it requires a careful configuration as too much gas lift will significantly increase the down hole pressure and result in a decrease in production flow instead.

$$Qoil = a_0 + a_1 whp + a_2 whp^2 + a_3 glr + a_4 glr^2 + a_5 whpglr + a_6 whp^2 glr + a_7 whpglr^2 \quad (2)$$

6.1.2. Test Separator Data

The number of samples in the test separator data are few in comparison to the collected field data. The oil, water, and liquid flow rates are measured using accurate multiphase flow meters but at an infrequent periodicity determined by the operators discretion. The test separator data further includes relevant averaged field measurements, which are well head pressure, choke percentage, gas lift rate, down hole pressure and well head temperature.

6.1.3. MIKON Estimated Flow Rates

MIKON is a specially designed first-order polynomial model in predicting oil flow rates, Qoil, and can be seen as a blend of both a hydrodynamical model and data driven VFM. The prediction of the water flow rate, Qwater, is found by multiplying the predicted oil flow rate with a weighting based on the wells water cut ratio, wtc (Equation 1).

$$Qwater = Qoil \times \frac{100}{100 - wtc} \quad (1)$$

Thereafter, the liquid flow rate can be predicted by taking the summation of the oil and water flow rate predictions. The MIKON model is essentially a linear regression model whose features are designed based on the hydrodynamical understanding of well flows. The model is updated on a periodic basis when test separator measurements are collected. The equation for the MIKON model is shown in equation 2. The MIKON model predictions are formerly known to have a 15% uncertainty range, regardless, the periodic tuning of the model ensures the predicted values estimate closely to the separator flow rate measurements.

6.1.4. Recorded Interventions

A list of recorded man-made interventions throughout the wells lifetime is provided. From inspection, the majority of well interventions are choke changes due to sanding. Furthermore, the majority of recorded interventions date between 2016 through to the end of 2017, and from visualisation of the field data, as seen in Figure 2, there are clearly other well interventions that happen after 2016 but are not recorded. Because of this, the Recorded Interventions are disregarded from the investigation.

In light of this, it is known however that the most significant interventions are due to acid scale treatment which removed the build up of calcium carbonate scaling. This gradual build-up of scale and scale removal can clearly be seen from the visualisation of the well characteristics, especially in liquid flow rate, as shown in Figure 3. The recorded periods of such scale treatments date from something to something.

6.2. Data Cleaning

The mlflowrate software is first used to clean and analyse the aforementioned types of relevant data sources. Initially, the provided field data comes in two separated data files in the format of the time, tag-name, and data columns. The tag-name contained two pieces of information for sorting, the first being the oil well from which the data originated, and the second being the sensor measurement type.

The appropriate process to deal with this is to first combine the two data files using the append_data() method. The data is then reformatted and organised into a grouped dictionary using the organise_data() method with the specified option, 'date_tag_val_col', and name code dictionaries are passed to further collect data into its respective wells and features. Another advantage to sorting into the grouped dictionary is that it can determine the number of samples collected for each sensor measurement. The number of samples, null, and duplicate values for the field data is displayed in Table 1.

The given field data is clearly unsuitable for predictive modelling. The functionality to clean such data is provided via the Data class' clean_data() method. The remove_null and avg_over arguments eliminate the problematic missing values and duplicated samples. Next, the inconsistency of samples across the features is corrected using the organise_data() method with the 'dict_col' option. This option calls the private method _dict2df() to reconstruct the grouped dictionary format back into a contiguous Spark DataFrame.

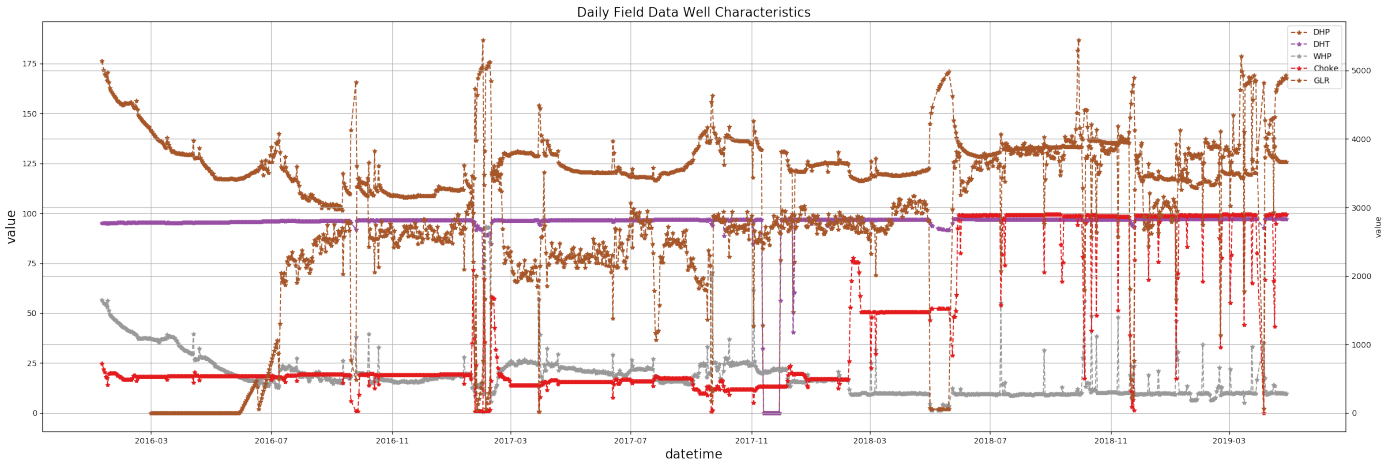

Figure 2: Visualisation of the oil well characteristic field data, the sudden peaks and drops to zero indicate some form of intervention

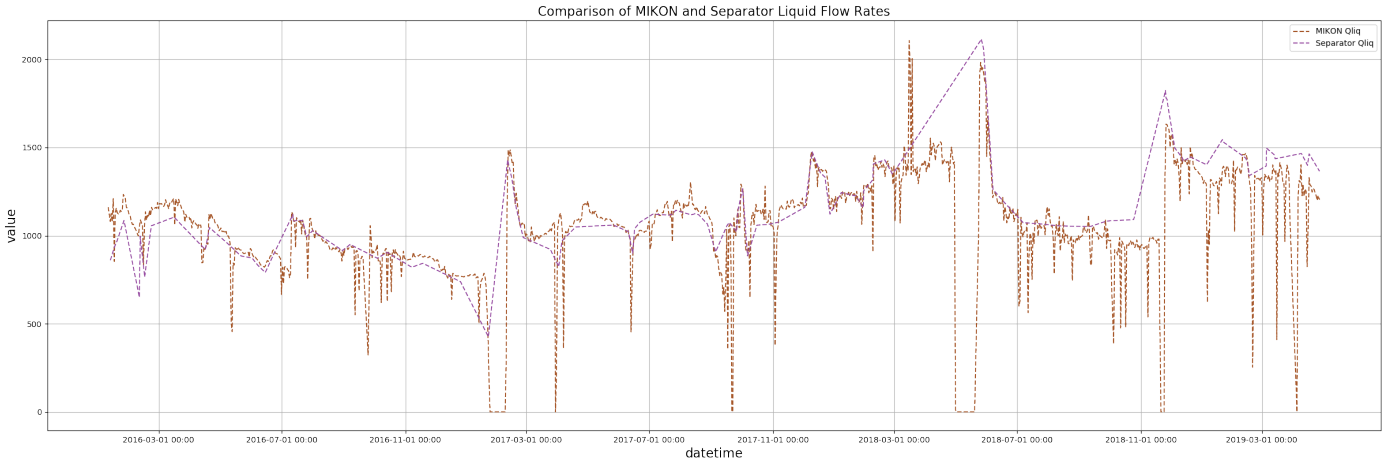

Figure 3: Visualisation of the liquid rate data, the sharp drop in liquid rates after 2018 indicate scaling

Feature	Samples	Nulls	Duplicates
wht	59086	344	6
whp	86448	0	10
dht	86448	56648	10
dhp	86448	56648	10
glr	59086	344	6
glp	30239	344	3
choke	59086	366	6
asd	64799	870	7

Table 1: Discontiguous features with missing values and duplicates in field data

Feature	Samples	Nulls	Duplicates
wht	1234	0	0
whp	1234	0	0
dht	1234	0	0
dhp	1234	0	0
glr	1234	0	0
glp	1234	0	0
choke	1234	0	0
asd	1234	0	0

Table 2: Cleaned field data via MLFlowRate Data tools

Tables 1 and 2 show the before and after cleaning the field data. As can be seen in table 2, the resulting clean majorly reduces the field data down to 1234 samples.

The next two data sources, MIKON data and Separator data, follow the same processes to data cleaning. The MIKON data source contains the oil, water, and liquid flow rate estimations, whereas the Separator data contains the flow rates measurements. Prob-

lematically, both data sources included a number of irrelevant features for miscellaneous engineering calculations. These features were filtered out via the select_col() and drop_col() methods. Both methods perform the same functionality, but are implemented in the case that only a few features are desired to be kept or removed, hence improving the user experience. Across all the data sources, the last form of data cleaning was to ensure that the feature names were consistent across our data.

6.3. Feature Selection

Given our data scenario, it is known that well shut-ins will most certainly have a zero flow rate measurement. Discussions with production engineers, confirmed that the MIKON model indicates a zero liquid flow rate when the well has shut-in with near 100% accuracy. This means that we can take the samples of periods when the MIKON model predicted zero liquid flow rates. Therefore, our set of labelled liquid rates can be significantly enhanced. To add, if the models were trained on this extra portion of data, the models would learn when a well shut-in has occurred. As seen in Figure 3, the original labelled liquid rate data shows no signs of well shut-ins.

In addition to moving samples to increase the amount of labelled data, we know that the features in the labelled data are the same as the features found in the unlabelled field set. Therefore, it may be valuable to create an extra labelled dataset with the additional features included. The additional features found in the field set but not in the separator set are down hole temperature, acoustic sand detection, and finally gas lift pressure. To decide which of the three features we should be included in our additional labelled set. We can plot the distribution of features to give us an indication for which feature to include (Figure 4).

It appears from the distributions that down hole temperature has a distribution whose samples lie mainly in the 100 range. This provides an indication that the feature would not be so helpful in modelling as its variance is small. We shall exclude this feature from our labelled dataset, and only transfer the gas lift pressure and acoustic sand detection features to the dataset.

Having built our two datasets for training models, we shall use the fitpredict_naivemodels() method to run a series of standard models that are commonly used in machine learning. The results on both datasets are presented and discussed in the next section.

6.3.1. Model Results

The nested k-fold cross validation with 10 k-folds for the outer loop, and 5 k-folds for the inner loop were used in evaluating the general error estimates. The model results trained on the dataset without any additional features is show in Table 3.

Model	R2	RMSE
MIKON	0.969	106.521
Linear Regression	0.780	265.338
ElasticNet	0.776	268.138
Lasso	0.793	258.953
Ridge	0.783	264.884
PLS Regression	0.779	266.237
Random Forests	0.881	174.538

Table 3: Only Sep features data

We see that there is a significant improvement in accuracy when using the non regressive model, Random Forests. All the regression models have an R2 score below 0.8. However, the Lasso model can be seen to perform the best out of the regression models. The results for the second dataset is shown in Table 4.

Model	R2	RMSE
MIKON	0.969	106.521
Linear Regression	0.786	261.442
Elastic Net	0.786	262.543
Lasso	0.787	261.622
Ridge	0.787	261.244
PLS Regression	0.782	264.510
Random Forests	0.853	191.913

Table 4: Entire field features without dht data

It is highlighted that the overall error metrics produce overall higher errors than the previous dataset. However, all the regression model errors have become very similar. It may be that by including the additional features, we provide the models a greater

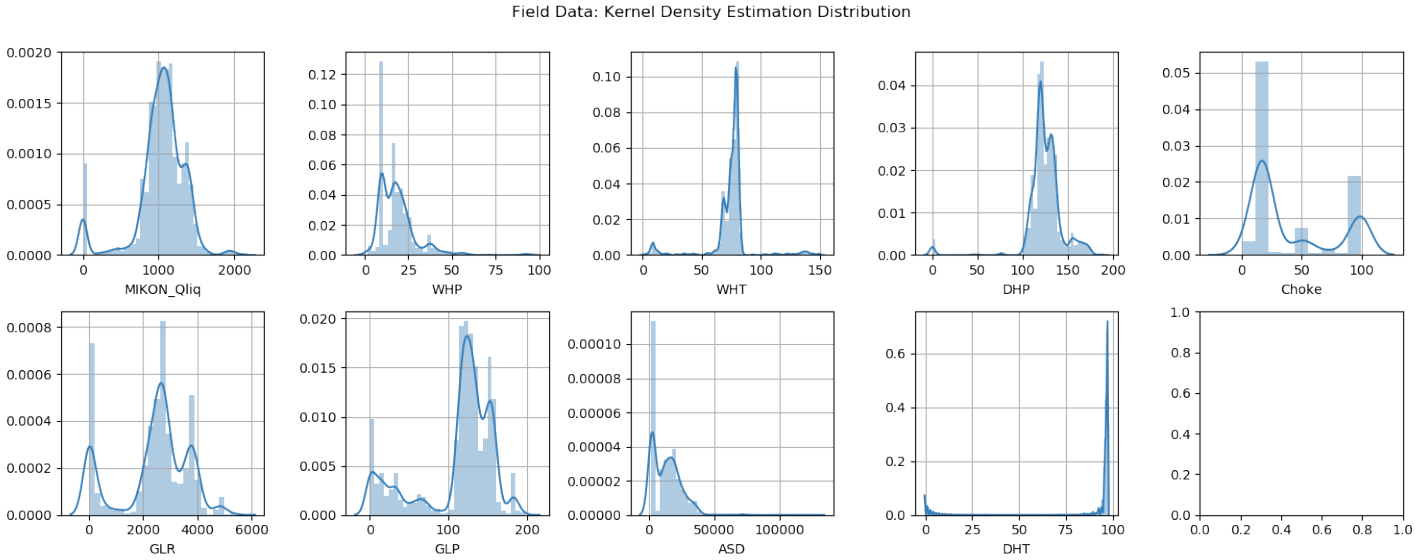

Figure 4: A kernel density distribution for the field data

feature space to train on, thereby increasing its sensitivity to unseen data and increasing model variance. In addition, the fact that the dataset is relatively small means that our models become more susceptible to overfitting. In both dataset cases, using random forests consistently outperforms the regression methods. This may be explained by the fact that the ensembling of decision trees selects random subsets of features for its trees, thereby ignoring the multi-colinearity of features. By plotting Pearson's correlation matrix we can see how much each feature is linearly correlated to one another as seen in Figure 5.

As the random forest model demonstrated the highest general error, we shall test it on the unlabelled field dataset with the fewer features using the evaluate() method. The result can be seen in Figure 6.

7. Discussions and Conclusions

With the mlflowrate software, we have successfully integrated disorganised well data to predicting liquid flow rates. The development of the software features to integrate the three columned data format with inconsistent samples proved to be the most challenging aspect for the software. This was made more difficult given the need for the software to be user friendly. Nevertheless, the ability to integrate data files for modelling liquid rates with an extensible software architecture have become the strengths of the software as demonstrated in the tools used for visualisation, cleaning, and assembly of data in the use case.

With regards to predictive modelling, it was found that using regression models produced relatively higher root mean squared errors ranging from 258 - 268, whereas the random forest model consistently had errors below 200. This is possibly due to the fact that the linear regression models suffered from the high multi-collinearity of features, whereas the random forest model is unaffected.

Although the model predictions did not perform better than the MIKON VFM. It is reasoned that no form of data preprocessing nor additional techniques were applied to the datasets. Nonetheless, this is one limitation of the software which needs to be addressed. Therefore, future works involve exploring more complex machine modelling techniques to integrate into the software. The factor analysis approach which reduces the dimensionality of features down to a number of latent dimensions will be the first exploration method for improving the predictions of the regression models. In addition, the use of autoencoders for pretraining neural networks in predicting liquid flow rate to leverage the large unlabelled dataset is another possibility for exploration.

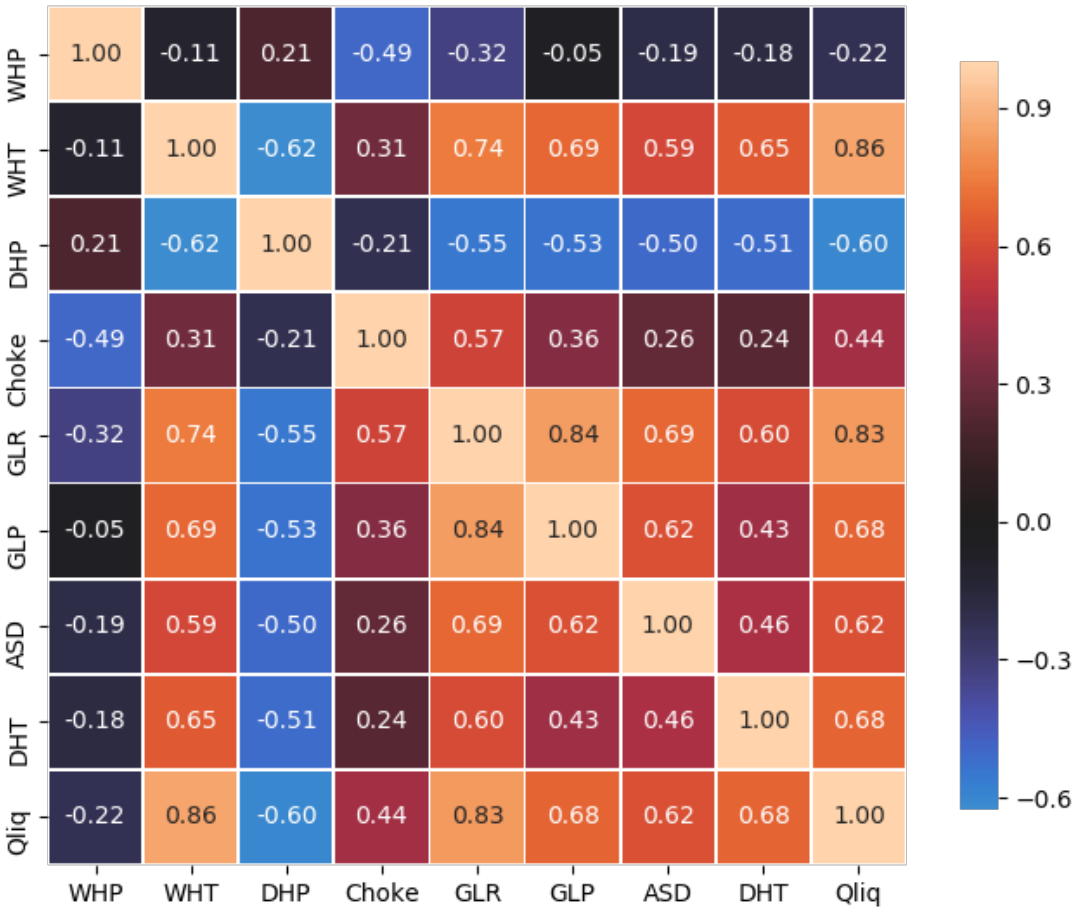

Figure 5: Pearson's Correlation of Features for Separator Data

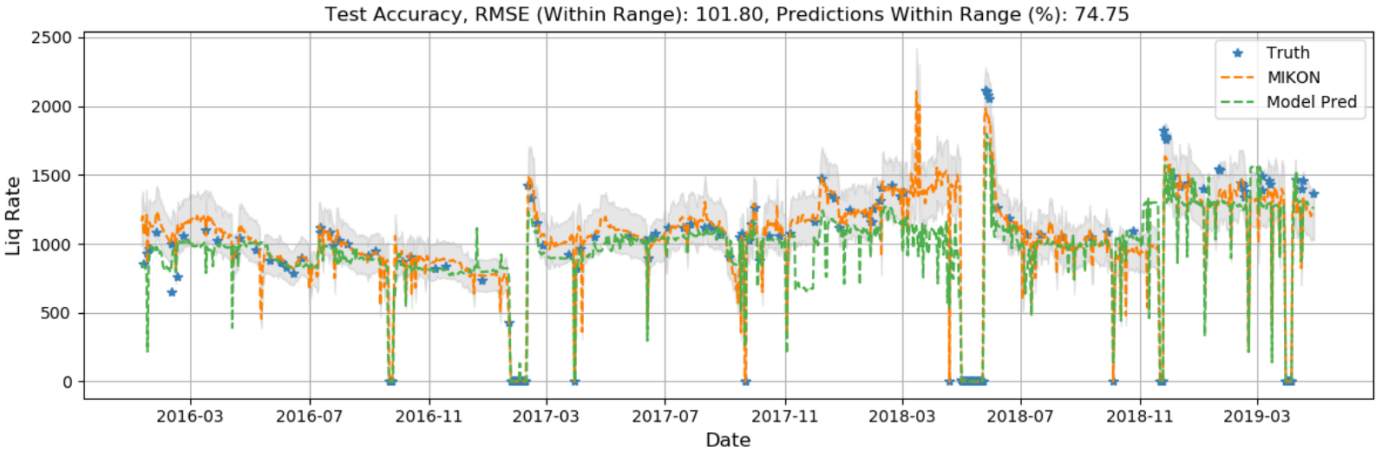

Figure 6: The evaluation test for the random forests can be seen to lie mostly inside the MIKON models uncertainty range

References

[1] M. C. . D. E. . P. F. . M. M. . A. J. . G. King, Fighting scale - removal and prevention, Oilfield Review (1999) 30–45.

[2] M. Zeinali Hasanvand, S. M. Berneti, Predicting oil flow rate due to multiphase flow meter by using an artificial neural network, Energy Sources, Part A: Recovery, Utilization, and Environmental Effects 37 (2015) 840–845. doi:10.1080/15567036.2011.590865.

[3] A. Amin, Evaluation of commercially available virtual flow meters (vfms), 2015. doi:10.4043/25764-MS.

[4] D. I. B. S. K. P. . M. E. Shoeibi Omrani, P., Improving the accuracy of virtual flow metering and back-allocation through machine learning, Society of Petroleum Engineers (2018) 30–45doi:10.2118/192819-MS.

[5] N. Andrianov, A machine learning approach for virtual flow metering and forecasting, IFAC-PapersOnLine 51 (8) (2018) 191 – 196, 3rd IFAC Workshop on Automatic Control in Offshore Oil and Gas Production OOGP 2018. doi:https://doi.org/10.1016/j.ifacol.2018.06.376.
URL http://www.sciencedirect.com/science/article/pii/S2405896318307067

[6] T. Ristanto, R. Horne, Machine learning applied to multiphase production problems., Ph.D. thesis (06 2018).

[7] S. Raschka, Model evaluation, model selection, and algorithm selection in machine learning, CoRR abs/1811.12808 (2018). arXiv:1811.12808.
URL http://arxiv.org/abs/1811.12808

[8] S. R. Varma S, Bias in error estimation when using cross-validation for model selection, BMC Bioinformatics 7 (91) (2006). doi:10.1186/1471-2105-7-91.