IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

PREVENTIVE ANALYSIS FOR APPLICATION SYSTEM FAILURE USING SERVER METRICS AND ENSEMBLE LEARNING METHOD

Self-learning solution to analyze the triggers, patterns and trends in historical logs and model to predict system failure.

[1]Venkata Yugesh Yerraguntla
[1]Software Engineer Sr Analyst
[1]Performance Engineering – Consumer Banking
[1]WellsFargo International Solutions Private Limited, Hyderabad, India.

Abstract: The purpose of this project is to build self-learning solution to analyze the triggers, patterns and trends in historical logs that resulted in prior system or application degradation / failure and to create a model to predict system or application failures based on real-time logs. In this scenario, the solution should be able to identify and predict the errors based on the real time log data and notify to the user or present the same in a readable format.

***Index Terms* - Application Monitoring, Error/anomaly Detection, Predictive Analysis, Random Forests, Supervised Learning, Server Log analysis.**

I. INTRODUCTION

Modern applications depend on functionality provided by other systems across the network. When these other systems experience performance problems or outages, the impacts can cascade across the many applications that are reliant upon them. For this reason, it is important to monitor these end points in order to better understand the scope of any impact and how it can be quickly mitigated to minimize any potential financial or reputational risk

We need to build self-learning solution to analyze the triggers, patterns and trends in historical logs that resulted in prior system or application degradation / failure and to create a model to predict system or application failures based on real-time logs. In this scenario, the solution should be able to identify a trigger for action, figure out what the rule should be, and reroute traffic from the failover instance. It can also examine criteria to recover the original environment. The goal of this project/paper is to present a solution that exhibits automated health checking with a resulting set of actions that demonstrates enhanced simplicity for the organization as well as technical simplification.

1.1 Objective:

- Identify and propose which existing machine learning models or customize one that can be deployed as a candidate for automation. Document your reasons for choosing your machine learning solution.
- Develop/utilize your ML model to connect to the empirical log artifacts. The Model should classify the data.
- Based on the classifications, look at the patterns that lead to the classification types. Predict with the real time data log the occurrence of classification type and execute prevent methods
- An email alert may be triggered based on the classifications
- For validation of the Model, Create two instances of a simple application that you will use as a primary instance and a failover instance.
- Create a UI that shows the user what incidents have been flagged by your ML solution and the resulting action(s).

1.2 Functional Scope of Work:

This project will automate the self-learning solution to analyse the triggers, patterns and trends in historical logs that resulted in prior system or application degradation / failure and to create a model to predict system or application failures based on real-time logs.

At a high level, the intended model will be used to follow the Predict, Prevent/Execute and Adapt process. The Predict will be based on empirical data and understand the classification, the Prevent/Execute process will be use case driven, the Adapt process will set the system to learn from the pattern. The expectation in this exercise will be a Prevent Model to ensure that some or all of the failures do not happen. From Execute perspective, the system will fire an action which will take an alternate path. The output of the exercise is to assist in simplifying the Prevent mechanism and should provide leads for Automations. This solution can be applied to integrated automated, predictive health checks and associated information can be incorporated into dashboards or live monitoring tools which can provide real time insights.

II. IMPLEMENTATION OVERVIEW

2.1 Product Context

This is a web-based application which is used for indicating the user with the incidents. This depends on the Machine Learning algorithm that we are using to predict the incidents based of historical data and utilize your ML model to connect to the empirical log artifacts. An email alert may be triggered based on the classifications. For ex, if one of the classifications is capacity issue, the ML solution should also trigger a failover/traffic re-route step to route traffic from the primary application instance to the failover instance

2.2 User Characteristics

In their current model of implementation, the response of these tools is reactive than preventive, mostly they report rather than respond, not anticipative rather static rules-based task-oriented systems, would heavily depend on human intervention & input, their recovery is isolated rather than achieving a self-recoverable overall business continuity. Functional dependencies are those where a project is dependent on another project for functionality. Non-functional dependencies are others, such as expectation that another project is creating a new namespace into which this project's changes will be placed.

2.3 System Characteristics

The system built will be an automated solution for predicting failure in systems or applications. Such a solution could be used to generate reports of suspicious behavior, substandard code or architecture, the triggering of real-time alarms to alert our support team(s), automatically initiating a fallback plan such as disaster planning/recovery, the spin-up of one or many failover/BCP environments, and shutting down a compromised point-of-entry.

2.4 Reliability

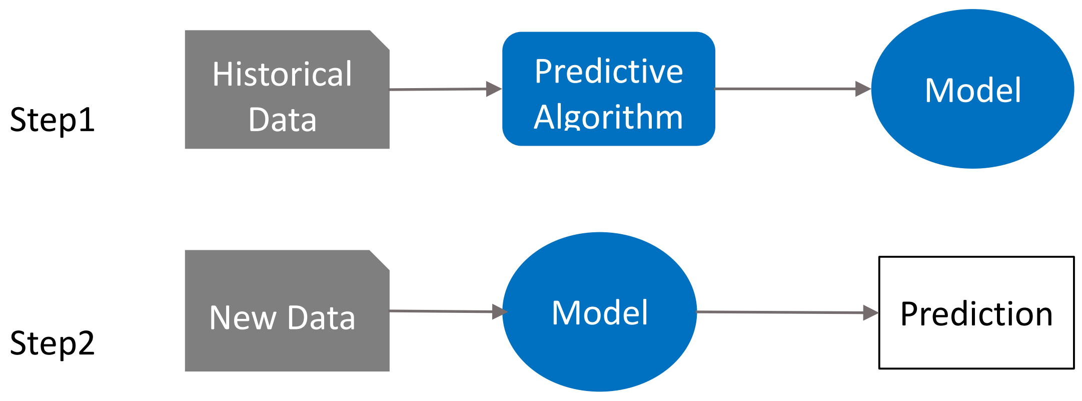
This software will be developed with machine learning, feature engineering and deep learning techniques. So, in this step there is no certain reliable percentage that is measurable. Also, user provided data will be used to compare with result and measure reliability. With recent machine learning techniques, user gained data should be enough for reliability if enough data is obtained.

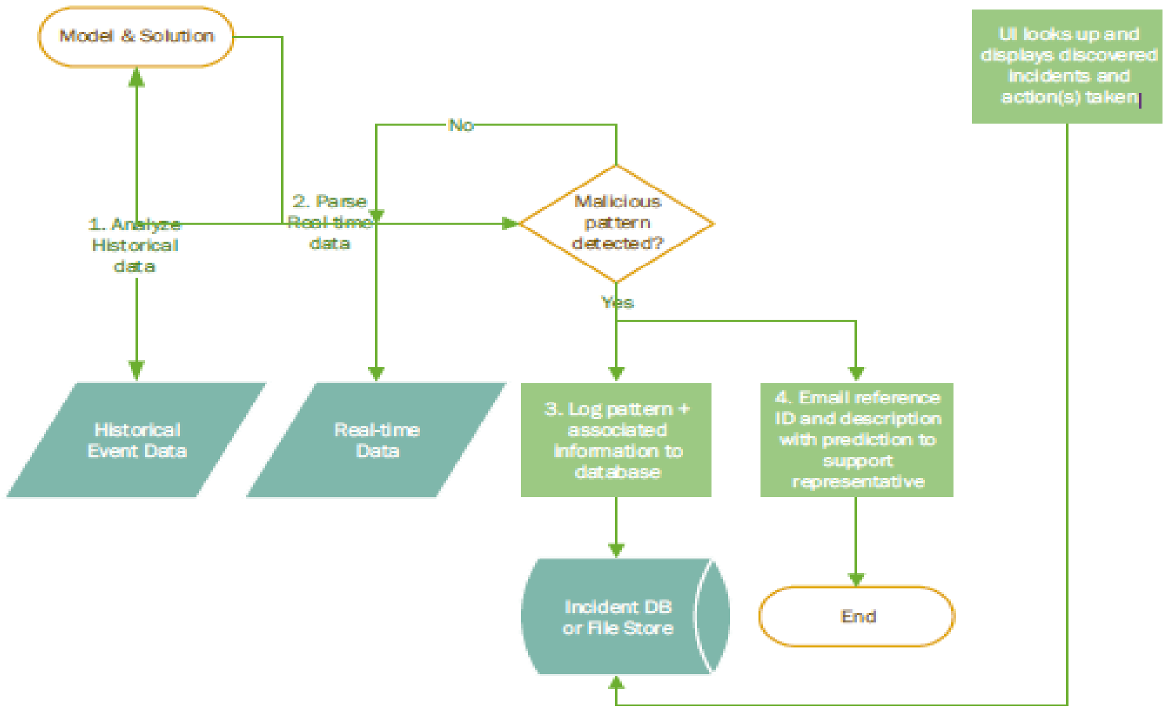
The maintenance period should not be a matter because the reliable version is always run on the server which allow users to access summarization. When admins want to update, it takes long as upload and update time of executable on server. The users can be reach and use program at any time, so maintenance should not be a big issue

III. IMPLEMENTATION APPROACH

Predictive modelling is a technique used to predict an event based on events happened in the past. It aims to work upon the provided information to reach an end conclusion after an event has been triggered.

We use Machine learning algorithms/models to derive the prediction, these models can belong to both supervised and unsupervised techniques and make use of classifiers and guess the probability of an outcome given a set of input data.




Model & Solution
UI looks up and displays discovered incidents and action(s) taken
No
1. Analyze Historical data
2. Parse Real-time data
Malicious pattern detected?
Yes
Historical Event Data
Real-time Data
3. Log pattern + associated information to database
4. Email reference ID and description with prediction to support representative
Incident DB or File Store
End

3.1 Data Collection:

A data set is a collection of data. In other words, a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. Supervised machine learning is the training of machine using a sample of labeled class data to train the machine what is right vs what is wrong. So after thousands to a millions sample of data, the machine gets to understand and observe patterns. On the other hand, unsupervised learning is to let the machine to learn on its own by trying to identify a pattern based on the provided data. The machine isn't told what data is useful vs what is not useful, nor which data is correct. In both of the above cases, the most important factor is not the learning process, but the quality of data.

Collecting the real time log data and parsing it for further analysis is the first step. We need to collect the raw log data from the servers and understand the data. In order to collect the raw data, we need to ensure if the system is up and running and then collecting the real time log data from servers. In order to collect the data from the servers, we can use data collection and analysis tools, cloud-based log management platforms. Data collected can be in structured, unstructured or semi structured format.

3.2 Data Conditioning/Data Preparation:

We need to prepare the collected data to a meaningful format. Active learning is useful when you have a large amount of unlabeled data and you need to decide what data you should label. Labeling data can be expensive, so we'd like to limit the time spent on this task.

General approach:

- Starting with an unlabeled dataset, build a "seed" dataset by acquiring labels for a small subset of instances
- Train initial model on the seed dataset
- Predict the labels of the remaining unlabeled observations
- Use the uncertainty of the model's predictions to prioritize the labeling of remaining observations

3.3 Data Validation:

Establish a single value optimization metric for the project. Can also include several other satisficing metrics (i.e., performance thresholds) to evaluate models, but can only optimize a single metric. Optimize for accuracy Prediction latency under some threshold. 90% coverage (model confidence exceeds required threshold to consider a prediction as valid). The optimization metric may be a weighted sum of many things which we care about. Revisit this metric as performance improves.

We also need to ensure to split the gathered log data into Training Data and Testing Data. The split can be 80% training data and 20% testing data or 70% training data and 30% testing data.

Once the data obtained is analyzed, we need to ensure to modify/ clean the data into understandable format. In this project, we performed log analysis through Excel Macros and Shell Scripting.

3.4 Model Evaluation:

Establish performance baselines on your problem. Baselines are useful for both establishing a lower bound of expected performance (simple model baseline) and establishing a target performance level (human baseline). Simple baselines include out-of-the-box scikit-learn models (i.e. logistic regression with default parameters) or even simple heuristics (always predict the majority class). Without these baselines, it's impossible to evaluate the value of added model complexity. If your problem is well-studied, search the literature to approximate a baseline based on published results for very similar tasks/datasets.

In general, Models are better at understanding number than the categorical data. Hence, we also need to ensure to convert the categorical data into numerical data using label encoding.

3.5 Model Refinement:

Once you have a general idea of successful model architectures and approaches for your problem, you should now spend much more focused effort on squeezing out performance gains from the model. Build a scalable data pipeline. By this point, you've determined which types of data are necessary for your model and you can now focus on engineering a performant pipeline. Apply the bias variance decomposition to determine next steps. Break down error into: irreducible error, avoidable bias (difference between train error and irreducible error), variance (difference between validation error and train error), and validation set overfitting (difference between test error and validation error).

If training on a (known) different distribution than what is available at test time, consider having two validation subsets: val-train and val-test. The difference between val-train error and val-test error is described by distribution shift.

3.6 Testing and Evaluation:

Different components of a ML product to test:

- Model evaluation aims to estimate the generalization accuracy of a model on future (unseen/out-of-sample) data. Training system processes raw data, runs experiments, manages results, stores weights.

Required tests:

- Test the full training pipeline (from raw data to trained model) to ensure that changes haven't been made upstream with respect to how data from our application is stored. These tests should be run nightly/weekly.
- Prediction system constructs the network, loads the stored weights, and makes predictions.

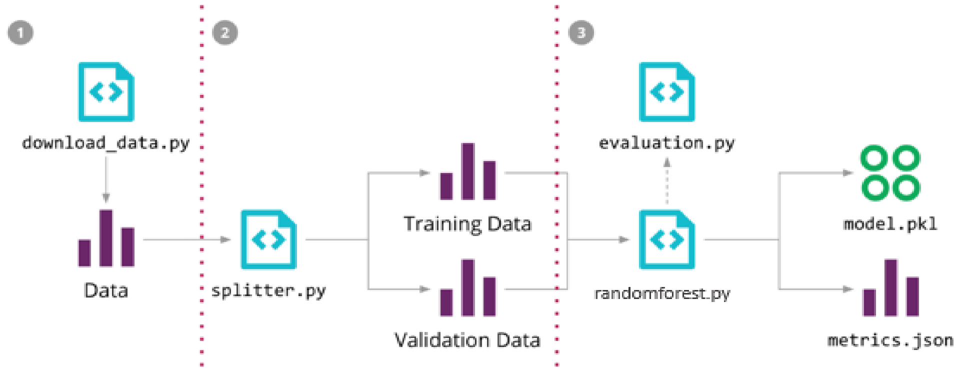
Required tests:

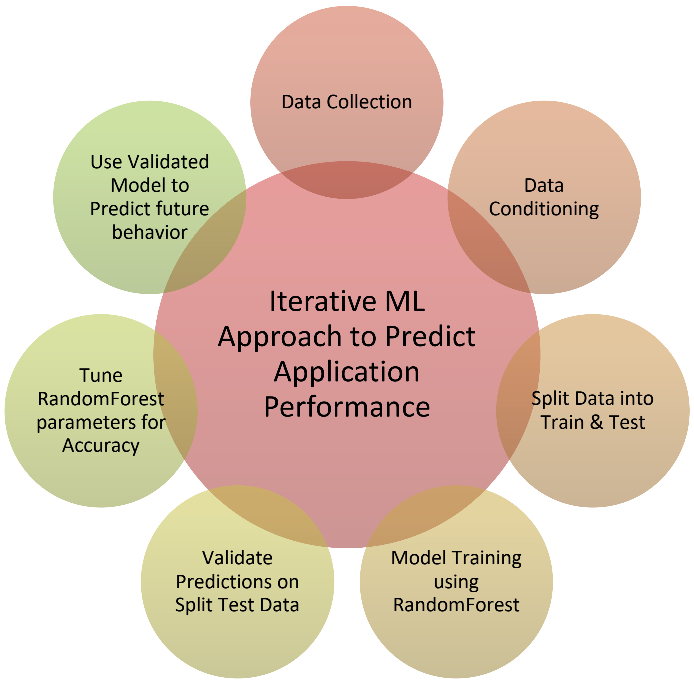
- Run inference on the validation data (already processed) and ensure model score does not degrade with new model/weights. This should be triggered every code push.

3.7 Model Deployment:

Model deployment be sure to have a versioning system in place for: Model parameters, Model configuration, Feature pipeline, Training dataset, Validation dataset.

Pipeline is an overloaded term, especially in ML applications. We want to define a "data pipeline" as the process that takes input data through a series of transformation stages, producing data as output. Both the input and output data can be fetched and stored in different locations, such as a database, a stream, a file, etc. The transformation stages are usually defined in code, although some ETL tools allow you to represent them in a graphical form. They can be executed either as a batch job, or as a long-running streaming application.



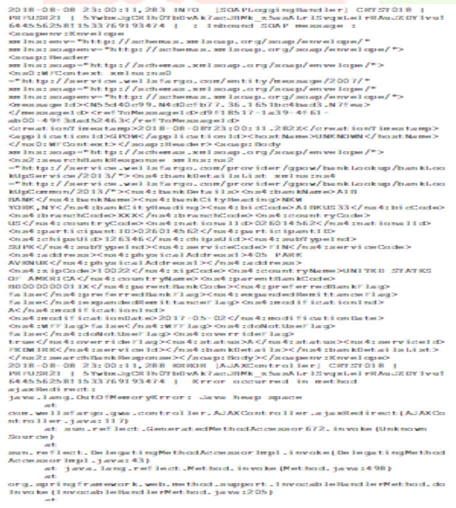

Iterative ML Approach to Predict Application Performance
Data Collection
Data Conditioning
Split Data into Train & Test
Model Training using RandomForest
Validate Predictions on Split Test Data
Tune RandomForest parameters for Accuracy
Use Validated Model to Predict future behavior

IV. Data Conditioning and Refinement

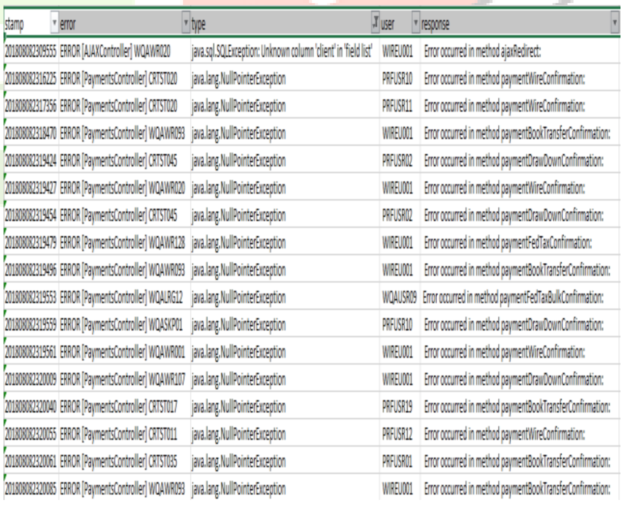
The main steps of Data preparation are: Sequence Extraction, Grouping and Classification and Removing Noise.

- Sequence Extraction: Input data of the failure predictor are error sequences. Each error event consists of a timestamp and a distinctive error name denoting the type of the error. As shown in above image, data is forming a sequence or errors.
- Grouping and classification: Training data needs to be extracted that should represent system characteristics as precisely as possible. We propose a method to group similar failure-related error sequences.
- Removing the noise: Error logs often contain events that are unrelated to a specific fault, but due to parallelism in the system these events are interweaved with unrelated events. This can be seen as noise in the data set.

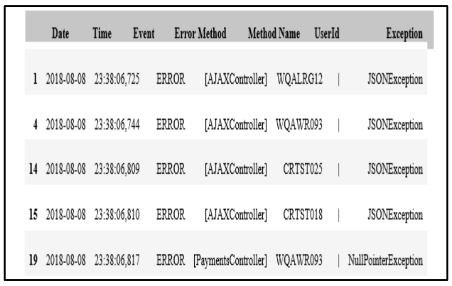
In the current project, as we are evaluating the server logs which are generally in unstructured format. In order to process this unstructured data format, we can use excel macros to separate the tags present in the log formats or shell scripts to parse the logs into meaningful fields.



```
2018-08-08 23:00:11,283 INFO  [SOAPLoggingHandler] CRTST018 |
PRFUSR21 | 5YwbxJgCR1h0TbDvAk7acJBMk_x5exALr15vgxLelrRAuJXDY1vu!
6445562581!1533769193474 |   : Inbound SOAP message :
<soapenv:Envelope
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<ns0:WFContext xmlns:ns0
="http://service.wellsfargo.com/entity/message/2007/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<messageId>CN55d40c99.N4d0cfb77.36.1651bc4bed3.N7fea>
</messageId><refToMessageId>d9f18517-1a39-4f61-
ab00-49f3dad52463</refToMessageId>
<creationTimestamp>2018-08-08T23:00:11.282Z</creationTimestamp>
<applicationId>GPOW</applicationId><hostName>UNKNOWN</hostName>
</ns0:WFContext></soap:Header><soap:Body
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<ns2:searchBankResponse xmlns:ns2
="http://service.wellsfargo.com/provider/gpow/bankLookup/bankLoo
kupService/2013/"><ns4:bankDetailsList xmlns:ns4
="http://service.wellsfargo.com/provider/gpow/bankLookup/bankLoo
kupCommon/2013/"><ns4:bankDetails><ns4:bankName>AIB
BANK</ns4:bankName><ns4:bankCityHeading>NEW
YORK,NY</ns4:bankCityHeading><ns4:bicCode>AIBKUS33</ns4:bicCode>
<ns4:branchCode>XXX</ns4:branchCode><ns4:countryCode>
US</ns4:countryCode><ns4:nationalId>026014562</ns4:nationalId>
<ns4:participantID>026014562</ns4:participantID>
<ns4:chipsUid>126346</ns4:chipsUid><ns4:subTypeInd>
SUPK</ns4:subTypeInd><ns4:serviceCode>FIN</ns4:serviceCode>
<ns4:address><ns4:physicalAddress1>405 PARK
AVENUE</ns4:physicalAddress1></ns4:address>
<ns4:zipCode>10022</ns4:zipCode><ns4:countryName>UNITED STATES
OF AMERICA</ns4:countryName><ns4:parentBankCode>
B000000001IX</ns4:parentBankCode><ns4:preferredBankFlag>
false</ns4:preferredBankFlag><ns4:expandedRemittanceFlag>
false</ns4:expandedRemittanceFlag><ns4:modificationInd>
A</ns4:modificationInd>
<ns4:modificationDate>2017-05-02</ns4:modificationDate>
<ns4:WFFlag>false</ns4:WFFlag><ns4:doNotUseFlag>
false</ns4:doNotUseFlag><ns4:overrideFlag>
true</ns4:overrideFlag><ns4:status>A</ns4:status><ns4:serviceId>
FKDWIRE</ns4:serviceId></ns4:bankDetails></ns4:bankDetailsList>
</ns2:searchBankResponse></soap:Body></soapenv:Envelope>
2018-08-08 23:00:11,288 ERROR [AJAXController] CRTST018 |
PRFUSR21 | 5YwbxJgCR1h0TbDvAk7acJBMk_x5exALr15vgxLelrRAuJXDY1vu!
6445562581!1533769193474 |  Error occurred in method
ajaxRedirect:
java.lang.OutOfMemoryError: Java heap space
        at
com.wellsfargo.gws.controller.AJAXController.ajaxRedirect(AJAXCo
ntroller.java:117)
        at sun.reflect.GeneratedMethodAccessor672.invoke(Unknown
Source)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethod
AccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
org.springframework.web.method.support.InvocableHandlerMethod.do
Invoke(InvocableHandlerMethod.java:205)
        at
```



stamp	error	type	user	response
201808082309555	ERROR [AJAXController] WQAWR020	java.sql.SQLException: Unknown column 'client' in 'field list'	WIREU001	Error occurred in method ajaxRedirect:
201808082316225	ERROR [PaymentsController] CRTST020	java.lang.NullPointerException	PRFUSR10	Error occurred in method paymentWireConfirmation:
201808082317356	ERROR [PaymentsController] CRTST020	java.lang.NullPointerException	PRFUSR11	Error occurred in method paymentWireConfirmation:
201808082318470	ERROR [PaymentsController] WQAWR093	java.lang.NullPointerException	WIREU001	Error occurred in method paymentBookTransferConfirmation:
201808082319424	ERROR [PaymentsController] CRTST045	java.lang.NullPointerException	PRFUSR02	Error occurred in method paymentDrawDownConfirmation:
201808082319427	ERROR [PaymentsController] WQAWR020	java.lang.NullPointerException	WIREU001	Error occurred in method paymentWireConfirmation:
201808082319454	ERROR [PaymentsController] CRTST045	java.lang.NullPointerException	PRFUSR02	Error occurred in method paymentDrawDownConfirmation:
201808082319479	ERROR [PaymentsController] WQAWR128	java.lang.NullPointerException	WIREU001	Error occurred in method paymentFedTaxConfirmation:
201808082319496	ERROR [PaymentsController] WQAWR093	java.lang.NullPointerException	WIREU001	Error occurred in method paymentBookTransferConfirmation:
201808082319553	ERROR [PaymentsController] WQALRG12	java.lang.NullPointerException	WQAUSR09	Error occurred in method paymentFedTaxBulkConfirmation:
201808082319559	ERROR [PaymentsController] WQASKP01	java.lang.NullPointerException	PRFUSR10	Error occurred in method paymentDrawDownConfirmation:
201808082319561	ERROR [PaymentsController] WQAWR001	java.lang.NullPointerException	WIREU001	Error occurred in method paymentWireConfirmation:
201808082320009	ERROR [PaymentsController] WQAWR107	java.lang.NullPointerException	WIREU001	Error occurred in method paymentDrawDownConfirmation:
201808082320040	ERROR [PaymentsController] CRTST017	java.lang.NullPointerException	PRFUSR19	Error occurred in method paymentBookTransferConfirmation:
201808082320055	ERROR [PaymentsController] CRTST011	java.lang.NullPointerException	PRFUSR12	Error occurred in method paymentWireConfirmation:
201808082320061	ERROR [PaymentsController] CRTST035	java.lang.NullPointerException	PRFUSR01	Error occurred in method paymentBookTransferConfirmation:
201808082320085	ERROR [PaymentsController] WQAWR093	java.lang.NullPointerException	WIREU001	Error occurred in method paymentBookTransferConfirmation:



	Date	Time	Event	Error Method	Method Name	UserId	Exception
1	2018-08-08	23:38:06,725	ERROR	[AJAXController]	WQALRG12	|	JSONException
4	2018-08-08	23:38:06,744	ERROR	[AJAXController]	WQAWR093	|	JSONException
14	2018-08-08	23:38:06,809	ERROR	[AJAXController]	CRTST025	|	JSONException
15	2018-08-08	23:38:06,810	ERROR	[AJAXController]	CRTST018	|	JSONException
19	2018-08-08	23:38:06,817	ERROR	[PaymentsController]	WQAWR093	|	NullPointerException

V. MODEL EVALUATION

Machine Learning has touched every aspect of our lives in the recent past and is transforming the way problems are solved in every imaginable field, in ways more than we realize. Ranging from transportation, cancer-detection, surveillance/crime detection to bio-informatics, recommendation engines of e-commerce websites and digital voice assistants around us, ML and AI are solving problems today which were considered "too big" to tackle earlier and are continuing to evolve at a rapid pace.

Choosing the appropriate algorithm:

The given problem of predicting server failure based on server logs can be approached using supervised learning methods of classification. Since, we have multiple target labels (error types in this case), we would need a Multiclass model over a binary one to cover all possible classes for prediction. In the field of machine learning there are various algorithms in use for such modelling, based on the dataset size, complexity and distribution of various classes. The most popular classification algorithms being logistic classification, decision trees, random forests and convolutional neural networks.

For the dataset at hand for this problem, we had close to 5 gigabytes of logs which when parsed converted to roughly 0.5 million unique records. However, when we filtered this for the data relevant to problem statement, we got roughly 73,000 unique records.

We have tried the following three algorithms for this problem on scikit learn in Python and compared their accuracy and sensitivity using Confusion Matrix and Classification Report:

1. Logistic Classification:

We observed a very fast convergence of the training model using lgbs solver and a balanced class weight for multiclass classification. The model performed poorly on validation as all the target labels were marked under the same unique class, totally ignoring the other three classes present in the training data. Hence this model was rejected. Another disadvantage is its high reliance on a proper presentation of your data. This means that logistic regression is not a useful tool unless you have already identified all the important independent variables. Since its outcome is discrete, Logistic Regression can only predict a categorical outcome. It is also an Algorithm that is known for its vulnerability to overfitting.

2. Decision Tree Classifier:

Decision trees are known to be more efficient in classification problems and we observed an increase in the number of predicted system failures overall and also improved sensitivity over logistic classification method. A small change in the data can cause a large change in the structure of the decision tree causing instability. For a Decision tree sometimes calculation can go far more complex compared to other algorithms. Decision Tree algorithm is inadequate for applying regression and predicting continuous values.

3. Random Forest Classifier:

Random Forests are another way to extract information from a set of data. The appeals of this type of model are:

It emphasizes feature selection — weighs certain features as more important than others.

It does not assume that the model has a linear relationship — like regression models do.

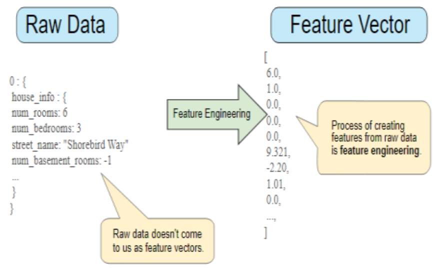
It utilizes ensemble learning. If we were to use just 1 decision tree, we wouldn't be using ensemble learning. A random forest takes random samples, forms many decision trees, and then averages out the leaf nodes to get a clearer model.

Random forest algorithm which adds bagging technique to decision tree approach is known for its ability to give high accuracy predictions even on limited size datasets. We used RF with k-fold GridSearchCV techniques and found the classification predictions to be the most accurate and sensitive. We zeroed in Random Forest Classification for this problem based on the accuracy observed.

In future, we can also try SVM and CNN techniques for this problem and evaluate if there are any accuracy or performance gains.

Predictive Model: A system of combination of various algorithms to cluster and classify the logs passed by Data Cleansing step and predict the output class and severity of it.

Label Encoding: Machines are good at understanding number than categorical data hence we convert categorical data into numerical using Label encoding. After applying the text analysis techniques, logs are converted into vector of numbers which computers can interpret. This technique of converting a group of words into numerical vectors is called Feature Generation.


Raw Data
Feature Vector
Feature Engineering
Process of creating features from raw data is feature engineering.
Raw data doesn't come to us as feature vectors.

```
0 : {
 house_info : {
 num_rooms: 6
 num_bedrooms: 3
 street_name: "Shorebird Way"
 num_basement_rooms: -1
 ...
 }
}
```

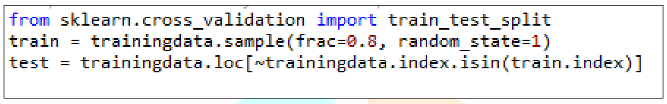
```
[
 6.0,
 1.0,
 0.0,
 0.0,
 0.0,
 9.321,
 -2.20,
 1.01,
 0.0,
 ...,
]
```

For the purpose of this study, we have used Python 3.6 as the primary programming language and the following packages/utilities for specialized functions:

- Language: Python 3.6.3
- IDE: Spyder 3.2.4 on Anaconda 5.0.1 64 bit distribution
- Pandas library for data analysis
- Scikit-learn library for regression modelling
- Random Forest Algorithm for training and prediction.
- numpy for Mathematical functions
- matplotlib, pydotplus, seaborn for visualization
- graphviz 2.38.0 for visualization
- MS Excel 2013 for curve plotting

5.1 Method and Observation:

The model under consideration here is meant for predicting error or faults in production environment, based on the corresponding observed transaction errors or issues in test environment, error time stamp, error description, user concurrency and application server response. These metrics were acquired from the log analysis (logs considered from the system through either a load test or through streaming). We selected a middleware application to achieve a proper one to one mapping for production and test environment synchronization. The dataset used for training this model consisted of 73000 records with 4 features and 1 target label. For validating the prediction model, we performed a random 80:20 split on the training data.



```
from sklearn.cross_validation import train_test_split
train = trainingdata.sample(frac=0.8, random_state=1)
test = trainingdata.loc[~trainingdata.index.isin(train.index)]
```

5.2 Training the Model:

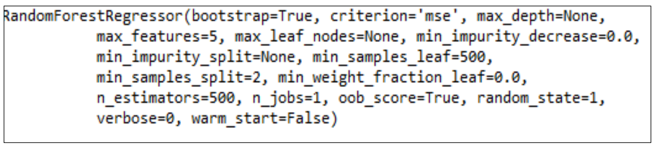
Before training, we performed a correlation analysis of the different features in our dataset to assess how strongly each of them correlate to the target label i.e., production response times. Dimensionality Reduction and PCA (Principal Component Analysis) were not performed for this study because of the limited number of features available in the training dataset.

Features Target Labels	• Error Time Stamp • Error Description • User Id • Server Response • Error Types
Sample Size (Training Data /Test Data)	~73k / ~70k
Training : Validation	80 : 20
Classification Accuracy	0.97 (imbalanced dataset)
Cohen Kappa Score	0.49 (recommended metric for validation)

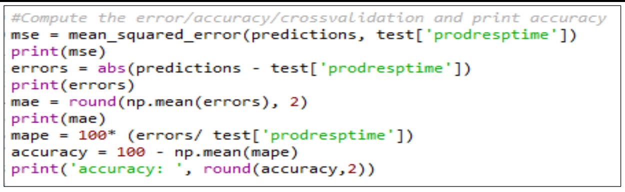
As a part of this study, we have developed and trained prediction models by mapping historic test metrics with historic production metrics over various releases which were available to us from the production logs and then used the trained model to predict production behavior for future releases using “Supervised Learning” methods, considering this as a regression problem.

We have used “Ensemble Modelling” technique of Random Forest to develop this non-linear model and tuned the hyper-parameters for maximum accuracy. Random Forest is a “Supervised Learning” algorithm which can be used for both classification and regression problems. It builds multiple decision trees and merges them together to get an accurate prediction by a technique known as bagging. Unlike Neural Networks, this algorithm is known to achieve reasonable prediction accuracy even with limited size of data-sets, as is the case with our data-set, limited by the number of test executions.

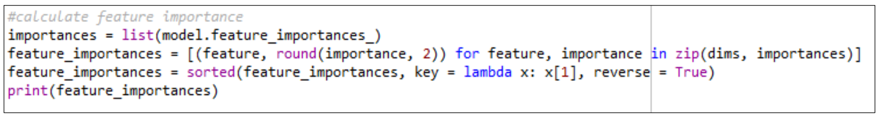
We used RF with k-fold GridSearchCV techniques and found the RF classification predictions to be the most accurate of all the methods tried (Logistic Classification, Decision Trees and RF).



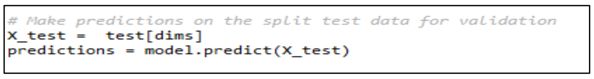
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=5, max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=500,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=500, n_jobs=1, oob_score=True, random_state=1,
           verbose=0, warm_start=False)
```



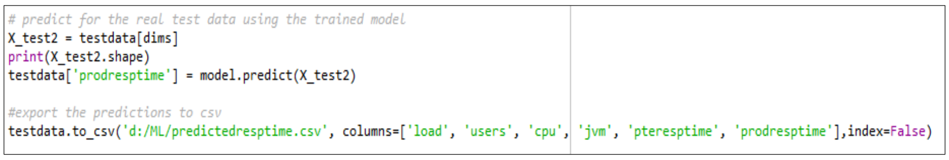
```
#Compute the error/accuracy/crossvalidation and print accuracy
mse = mean_squared_error(predictions, test['prodresptime'])
print(mse)
errors = abs(predictions - test['prodresptime'])
print(errors)
mae = round(np.mean(errors), 2)
print(mae)
mape = 100* (errors/ test['prodresptime'])
accuracy = 100 - np.mean(mape)
print('accuracy: ', round(accuracy,2))
```



```
#calculate feature importance
importances = list(model.feature_importances_)
feature_importances = [(feature, round(importance, 2)) for feature, importance in zip(dims, importances)]
feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse = True)
print(feature_importances)
```



```
# Make predictions on the split test data for validation
X_test =  test[dims]
predictions = model.predict(X_test)
```



```
# predict for the real test data using the trained model
X_test2 = testdata[dims]
print(X_test2.shape)
testdata['prodresptime'] = model.predict(X_test2)

#export the predictions to csv
testdata.to_csv('d:/ML/predictedresptime.csv', columns=['load', 'users', 'cpu', 'jvm', 'pteresptime', 'prodresptime'],index=False)
```

5.3 Discussion:

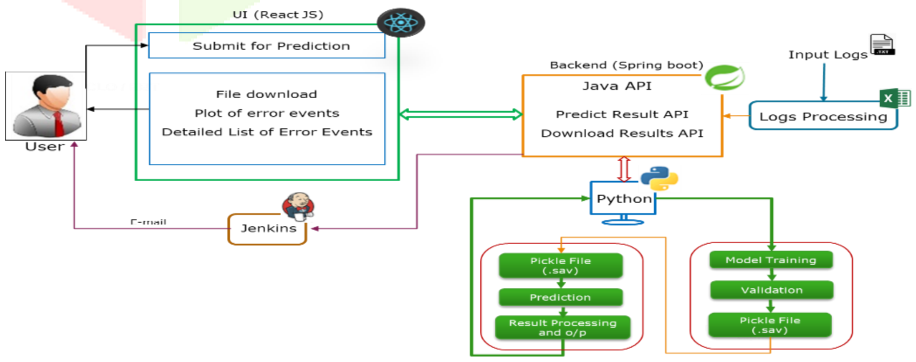
The proposed solution is thus shown to be helpful in predicting application performance in production using supervised learning techniques with reasonable accuracy. Before adopting this approach, it is important to understand the critical underlying assumptions.

Firstly, the modelling is based on the specific production and test environment size/configuration and their relative scaling factors which are usually very specific to each application at enterprise level. So, such models should not be used as a generic solution for different applications, and it is recommended to train unique models for each application for reasonable accuracy. Secondly, the parameters need to be tuned carefully for each model considering the fact that the optimum parameter values vary with dataset size and dispersion. This can be achieved iteratively while comparing the prediction accuracy on training data. For example, in the model under discussion, we observed that n_estimators parameter (number of trees) and min_sample_leaf parameter (minimum number of samples required to decide a node to be leaf) impacted accuracy while other parameters did not impact it much. Lastly, the model will be able to learn more about the system and make more accurate predictions, as we train it with more and more data. Hence, it is important to keep updating the model with more data points as and when available, to get closer to production behavior.

VI. SMART TOOL

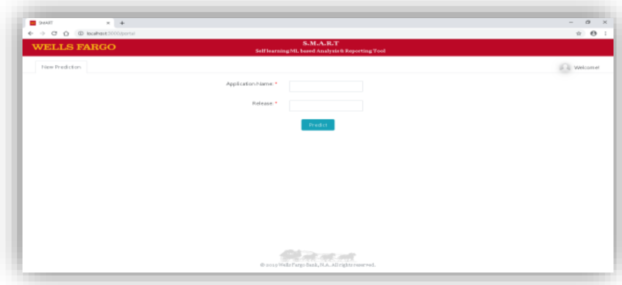
In order to present the results of the prediction and create an end-end monitoring tool, we have developed a web based tool to predict the system failures based on the real time logs.

Architecture Diagram:

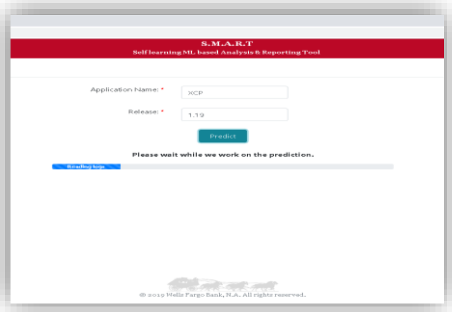


Tool stack:
React JS based User Interface
Java Spring boot Backend APIs

UI WIRE FRAME

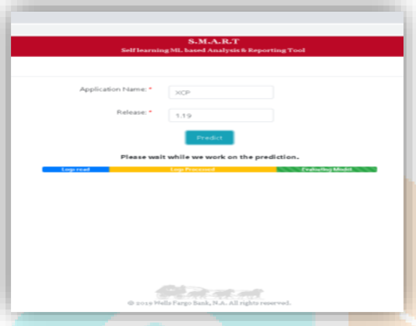

SMART
localhost:3000/portal
WELLS FARGO
S.M.A.R.T
Self learning ML based Analysis & Reporting Tool
New Prediction
Welcome!
Predict
© 2019 Wells Fargo Bank, N.A. All rights reserved.

Application Name: *
Release: *

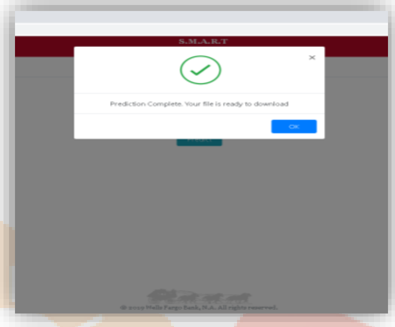

S.M.A.R.T
Self learning ML based Analysis & Reporting Tool
Predict
Reading logs
© 2019 Wells Fargo Bank, N.A. All rights reserved.

Application Name: * XCP
Release: * 1.19
Please wait while we work on the prediction.

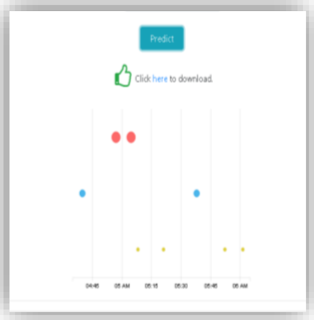
Enter Application Name, Release and Submit. These values would be used for extracting log files from various sources in real-time. Observe the progress states, Reading Logs, processing logs and evaluating model. Once the prediction is done, a popup would appear indicating the completion of prediction. Plot depicts the predicted error events with their corresponding timestamps.


S.M.A.R.T
Self learning ML based Analysis & Reporting Tool
Predict
Logs read
Logs Processed
Evaluating Model
© 2019 Wells Fargo Bank, N.A. All rights reserved.

Application Name: * XCP
Release: * 1.19
Please wait while we work on the prediction.



Prediction Complete. Your file is ready to download



SMART's UI is and independent Maven project based on REACT JS Framework.

Pre-requisites: IDE – Eclipse or equivalent**,** Maven**,** Node JS

How to execute locally?: Clone the repository on the local machine, Import "Smart" as an "Existing Maven Project", In the Terminal, execute the following commands - npm install, npm start.

Backend: Independent Spring Boot framework-based Maven Project. IDE – Eclipse or equivalent

VII. Summary

The presented model will be evaluating the server logs by considering them in real time and analyze the logs through a core model training engine. The model training engine is then coupled to validation and prediction system where we implement supervised training Random forest-based classifier. This will further be tuned with hyper parameter tuning which can be achieved with k-fold GridSearch Cross Validation. A React based UI is the frontend application which will trigger the python-based modelling and prediction engine. The output from the prediction engine, which is stored as a pickle file to make predictions on real time data. In order to visualize, the data will be presented as graphs and plot diagrams along with an exportable tabular format.

Differentiators:

- Highly adaptable and Lightweight solution which is easy to deploy
- Robust ML based solution, optimized for accuracy with k-fold Gradient Search Cross Validation Techniques.
- Real time visualization of the predictions on a React based UI.
- Automated Email alerts to respective teams using Jenkins.
- The trained model saved in a pickle file that can be used for every prediction.
- Loosely coupled architecture based on mostly open-source tools ensuring high scalability.

VIII. Conclusion

In the project, we've tried to leverage machine learning techniques to predict system failures based on historic data. We have also discussed the steps involved in building a prediction model from scratch and fine-tune it for accuracy, using open-source tools. The solution is easy to implement, customizable and low-cost for the problem it aims to solve. We can use similar approach to develop models to predict server metrics like CPU utilization, heap memory used etc. as an extension of this study. Such prediction models can be used to intelligently predict the system failures and errors in both legacy and cloud environments and thus, aid in smart monitoring at enterprise level.

IX. Acknowledgment

Thepreferredspellingoftheword "acknowledgment" inAmericaiswithoutan "e" afterthe "g".Avoidthestiltedexpression, "Oneofus(R.B.G.)thanks..."
Instead,try"R.B.G.thanks".Putapplicablesponsoracknowledgmentshere;DONOTplacethemonthefirstpageofyourpaperorasafootnote.

REFERENCES

[1] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
[2] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18--22.
[3] Travis E, Oliphant. A guide to NumPy, USA: Trelgol Publishing, (2006).
[4] Fernando Pérez and Brian E. Granger. IPython: A System for Interactive Scientific Computing, Computing in Science & Engineering, 9, 21-29 (2007), DOI:10.1109/MCSE.2007.53 (publisher link)
[5] Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)
[6] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55
[7] DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar School of Computing, University of Utah