

CAAP Quarterly Report

Date of Report: *July 6, 2020*

Prepared for: *Thomas Finch (Project Manager) and Joshua Arnold (CAAP Program Manager), U.S. DOT Pipeline and Hazardous Materials Safety Administration*

Contract Number: *693JK31850005CAAP*

Project Title: *Low-variance Deep Graph Learning for Predictive Pipeline Assessment with Interacting Threats*

Prepared by: *Hao Zhang (Colorado School of Mines) and Yiming Deng (Michigan State University)*

Contact Information: *Dr. Hao Zhang, Department of Computer Science, Colorado School of Mines; 1500 Illinois St., Golden, CO 80401; Phone: 303-273-3581; Email: h Zhang@mines.edu*

For quarterly period ending: *July 7, 2020*

Business and Activity Section

(a) Contract Activity

No contract modification was made or proposed in this quarterly period. No materials were purchased during this quarterly period.

(b) Status Update of Past Quarter Activities

(c) Cost Share Activity

PI Zhang used his effort as the 20% in-kind cost share to work on the project at the Colorado School of Mines. Co-PI Yiming Deng used effort as the 20% in-kind cost share to work on the project at the Michigan State University. The cost share was used following the approved proposal and no modification was made.

(d) Performed Research: Developing and Evaluating New Methods for Low-Variance Interacting Threats Assessment

1. Progress on Task 3

1.1. Complex Crack/Corrosion Geometry Design

Although defect of regular shape is being extensively studied and presented in the previous report, it suffers from the limitation that the overall inspection performance in our simulation is

usually “too good” to do reliability assessment since no topological variance or uncertainty is considered. This is a particular problem if the design of corrosion/crack geometry deviate from real scenarios. In COMSOL multiphysics, such design has been limited to regular 3D form or irregular 2D surface.

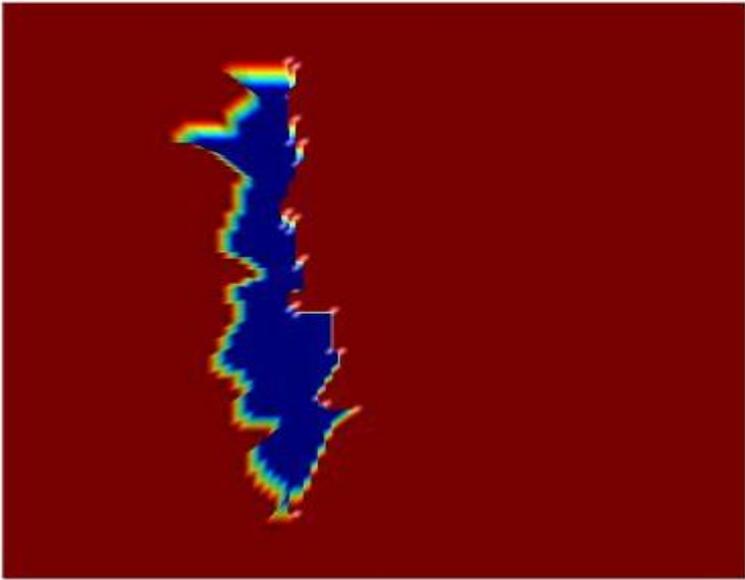


Figure 1: Overview of schematic crack design

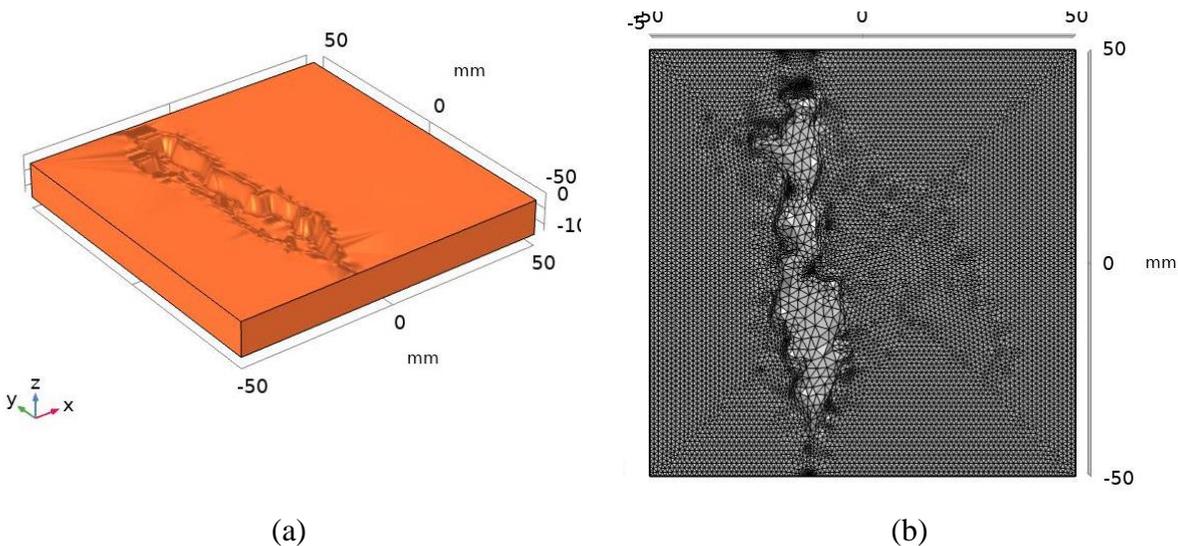
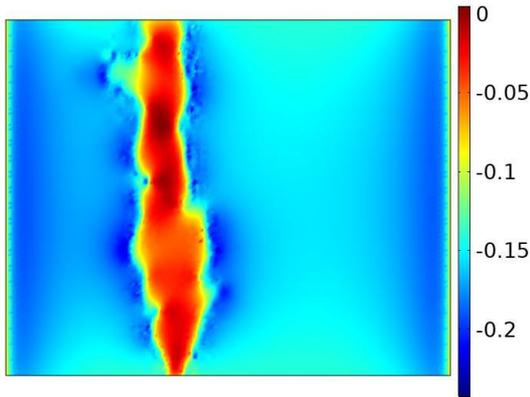


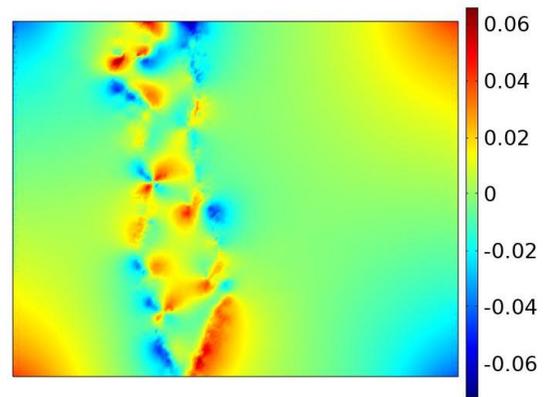
Figure 2: (a) Overview of crack geometry (b) Mesh plot of crack

The MATLAB-COMSOL link provides a possibility to demonstrate complex design with respect to topological irregularities, such as the insertion of zigzag opening at any place on top of

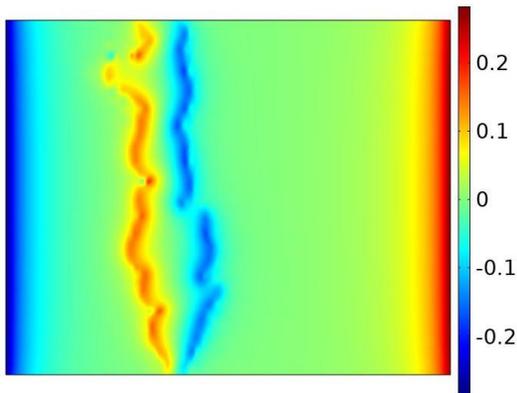
or inside the solid material. Topological sensitivity analysis, which originated from the classical shape information can be derived if we continue to simplify the model, generating sufficient data. Note that there are functions provided by COMSOL that could help us with the physics definition are limited. To construct irregular-shaped corrosion/crack geometry, actual values should be specified along x and y directions. Sources of crack/corrosion images can be found online where intensity value of each pixel could help us with estimating approximate shape. As the plot showing the Interpolation function of the imported file. The color bar values represent the depth of the crack/corrosion. To get a better representation, the Maximum number of knots and tolerance should be chosen wisely. For example, when knots number equals 100, this means that the specified area will be divided at most 100 pieces in both x and y directions, thus creating limited patches. The more knots that are allowed, the more flexibility is given to adjust the patches to the given z expression, thereby improving the chances of achieving a tighter relative tolerance. On the other hand, when relative tolerance is setup, the algorithm starts by dividing the whole area into a smaller number of patches and then increasing the number of patches where the error is large which lead to longer processing time as geometry become more complex.



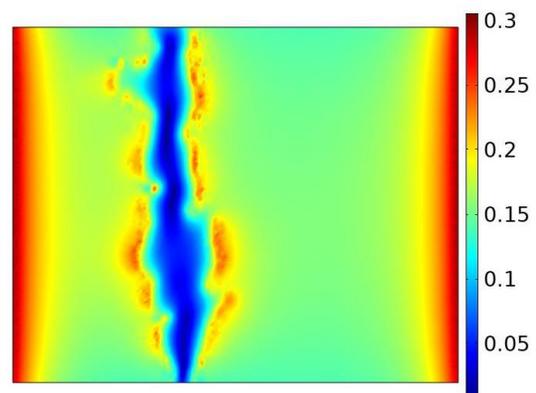
(a) Magnetic field B_x in [T]



(b) Magnetic field B_y in [T]



(c) Magnetic field B_z in [T]



(d) Magnetic field B_{norm} in [T]

1.2. Machine Learning Algorithms for Data Analysis

Here from the simulated data obtained from COMSOL we have extracted features. We have analyzed the features to understand the significant ones and at first perform regression to estimate the crack width. Here for regression we have considered various methods such as multiple linear regression, Polynomial regression, Support vector regression (SVR), Random Forest (RF) regression, Decision Tree regression (DT) and Artificial neural network (ANN) regression. Then we have evaluated the algorithms by the metrics such as R squared and Adjusted R squared metrics. Thus, here the crack width is the dependent variable which depends upon the various independent features (here 4). As the data acquisition from MFL inspection is not a burdensome method, hence the offline analysis and defect size reconstruction requires more sophisticated methods as the full reconstruction of a flaw is rarely possible. Hence here the degree of reliability comes into question. As the obtained features are nothing but various magnetic field values hence a mapping of magnetic fields onto flaw geometry results in an inverse problem. Here regression maps the crack width with different extracted features as the problem of mapping the MFL signal onto actual defect size is a regression problem. Different types of machine learning are considered and in the learning process the network is set up by minimizing the difference between the output of the artificial defects y_{pred} to their true values y_{true} . For the prediction the signal of an unknown defect is fed into the network and the output delivers an estimated defect size. Based on the various metrics discussed later we then evaluate the efficacy of different algorithms. Here we have done random train test split of the samples with 20% of the random data in the test set. Thus, if we have 41 data samples then test data contains 9 samples and the rest data going to train set. Here 4 features are independent while the crack width is the dependent feature to regress.

At first, we have applied regression by **multiple linear regression**. We have applied multiple linear as there are a number of variables upon which the crack width (y) depends. Here y is dependent on the features x_1, \dots, x_4 in a linear fashion. We have used the scikit learn api for applying the various regressors. From scikit learn we have imported the LinearRegression model and thereby applied regression by randomly portioning the data into train and test set. After applying the linear regression, the below table shows how close the predicted values are to the true ones. Here the precision is considered up to two places of decimals.

Table 1: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
6.7	7
9.04	9
7.81	7.80
2.8	2.8
3.96	4
8.28	8.2
7.26	7.4
4.17	4.2
9.23	9.2

Next, we have plotted the result of the true and predicted crack width against different features to show how accurate the results are.

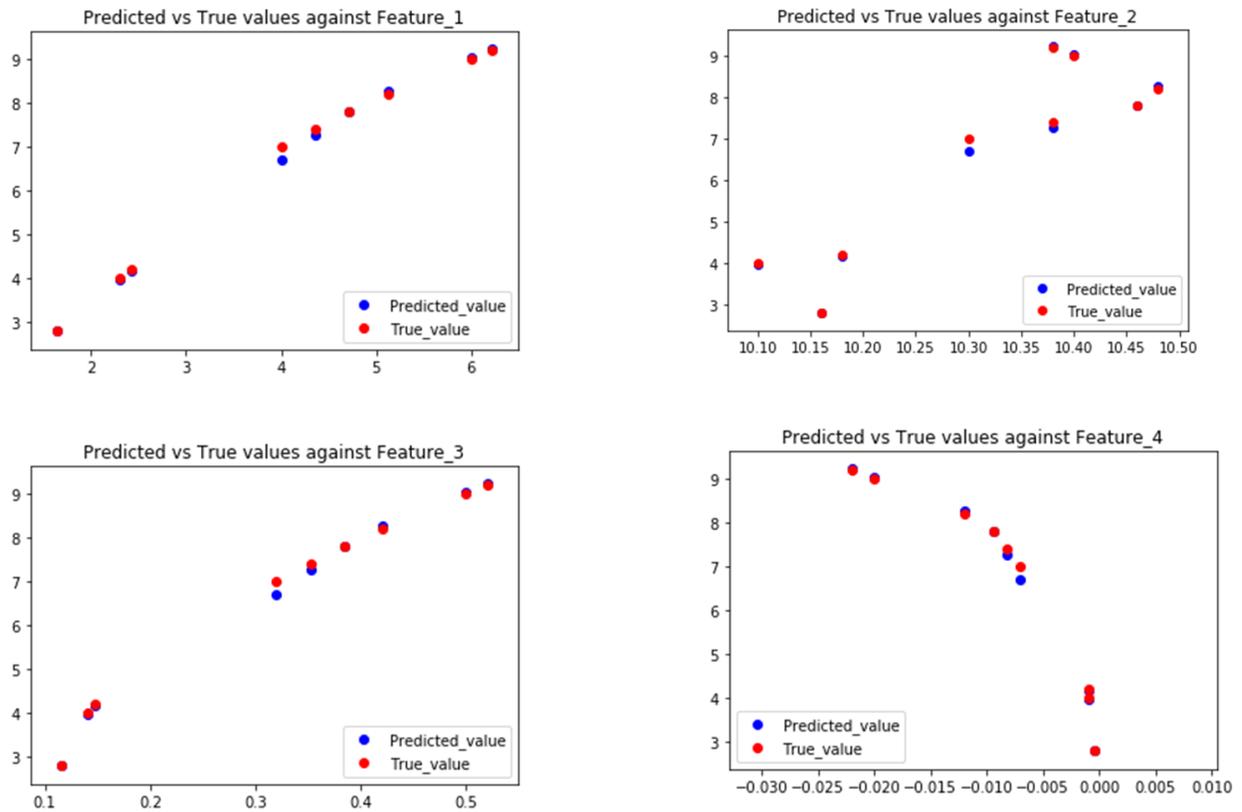


Figure 3: Predicted vs True values against different features

From the plots we can see the predicted values superimpose on the true values for all the features. The summary of the regressor shows the significant features based on the p values and the evaluated metrics like R-squared and multiple R-squared for the efficacy.

```
Call:
lm(formula = Crack_width ~ ., data = training_set)

Residuals:
    Min     1Q   Median     3Q    Max
-0.12502 -0.05088 -0.01103  0.02207  0.25650

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.09982    1.62414   1.909 0.066998 .
i..Feature_1  1.67992    0.09148  18.364 < 2e-16 ***
Feature_2   -0.34939    0.16190  -2.158 0.039976 *
Feature_3    4.80417    1.06924   4.493 0.000119 ***
```

```

Feature_4  144.94063  5.82169  24.897  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.08787 on 27 degrees of freedom
Multiple R-squared:  0.9989,    Adjusted R-squared:  0.9988
F-statistic: 6398 on 4 and 27 DF,  p-value: < 2.2e-16

```

Figure 4: Accuracy and the significant features based on p values

The P value and the significance value tells us the statistical significance of the dependent variables on the independent ones. From the significance column in the above figure it is clear that the feature 2 does not have so much significance as the other ones and hence do not affect the regression as other features and hence it's a redundant feature and can be omitted. Lower the p value more statistically significant that variable is. Feature 2 has significance between 1% and 5% and thus can be removed.

The metric **R squared** is given by:

$$SS_{res} = \sum (y_{true} - y_{pred})^2$$

$$SS_{tot} = \sum (y_{true} - y_{avg})^2$$

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

More the R value is close to 1, more accurate our result is. The accuracy score here is **0.9973** Next, we have applied **polynomial regression** on the same data set. To apply polynomial regression we have imported the PolynomialFeatures module from scikitlearn and then have implemented with degrees of freedom as 8. The below graph shows why the polynomial regression model is better than linear regression as it fits the data well.

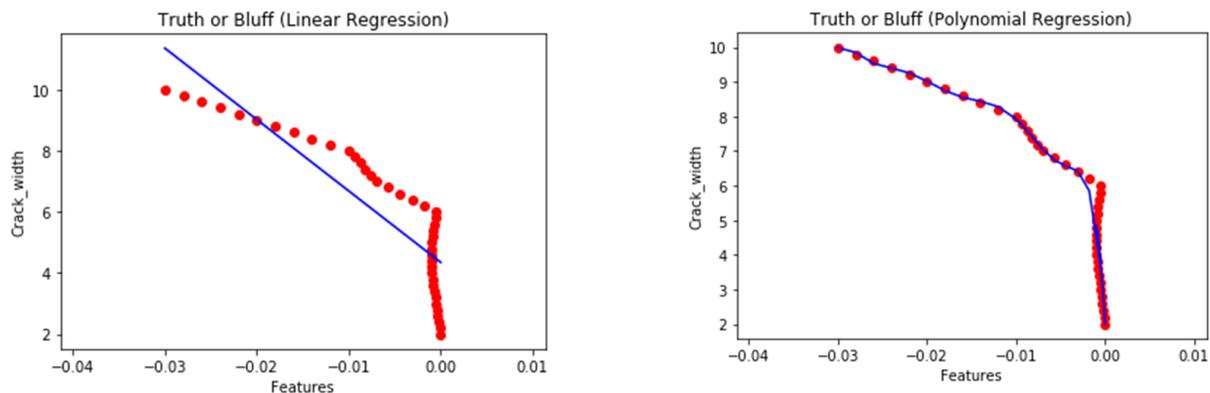


Figure 5: Comparison of linear vs polynomial regression

From the above figure it shows that the blue curve (obtained from regression) fits well with the red values (true ones) for polynomial regression. The table showing the true and predicted crack widths are as follows:

Table 2: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
7	7
9	9
7.8	7.80
2.8	2.8
4	4
8.2	8.2
7.4	7.4
4.2	4.2
9.2	9.2

Next, we have plotted the predicted vs true values for different features using polynomial regression to show how accurate this algorithm is. From the below figures we can see that the predicted values superimpose on the true values on all the cases thus making the R score as **0.999** which is almost 1.

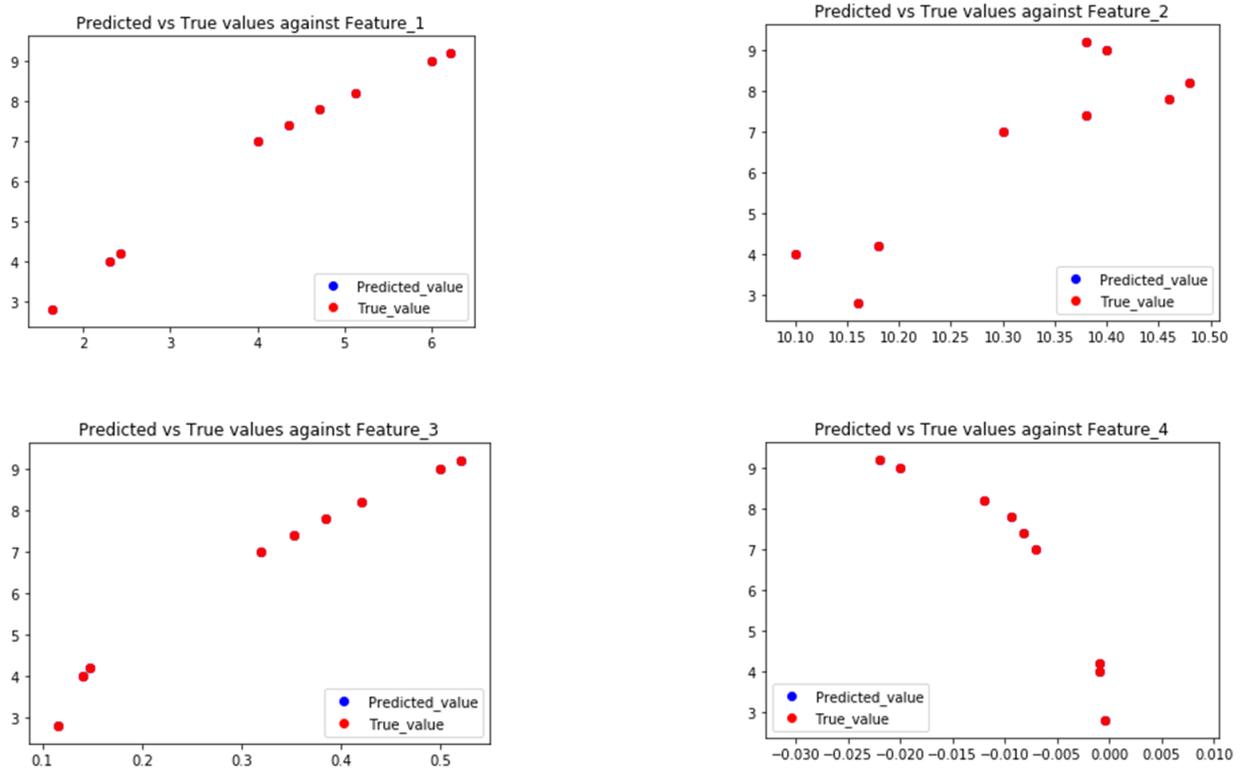


Figure 6: Predicted vs True values against different features

Next, we have implemented **Random Forest** for regression as it is one of the most powerful and accurate regression methodologies for nonlinear problems. However, the number of trees are to be chosen wisely. The RandomForestRegressor module is imported from scikit learn and is applied with number of estimators as 10. On applying regression on the entire dataset the below figure shows the blue line (predicted one) is close to the red (true one) for random forest regression.

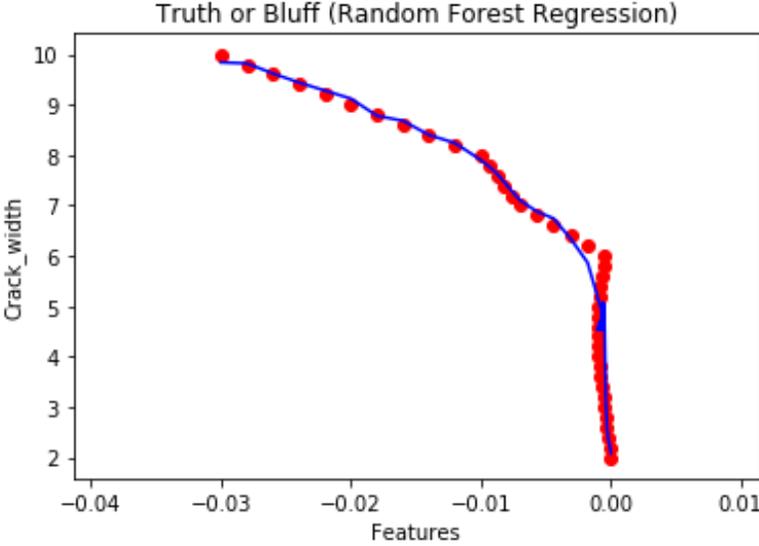


Figure 7: Random forest regression fitting the data

The table showing the true and predicted crack widths are as follows:

Table 3: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
6.96	7
8.82	9
7.64	7.80
2.84	2.8
3.76	4
8.22	8.2
7.44	7.4
4.24	4.2
9.46	9.2

Next, we have plotted the predicted vs true values for different features using random forest regression to show how accurate this algorithm is. From the below figures we can see that the predicted values are close to the true values not superimpose though, thus making the R score as **0.9957**.

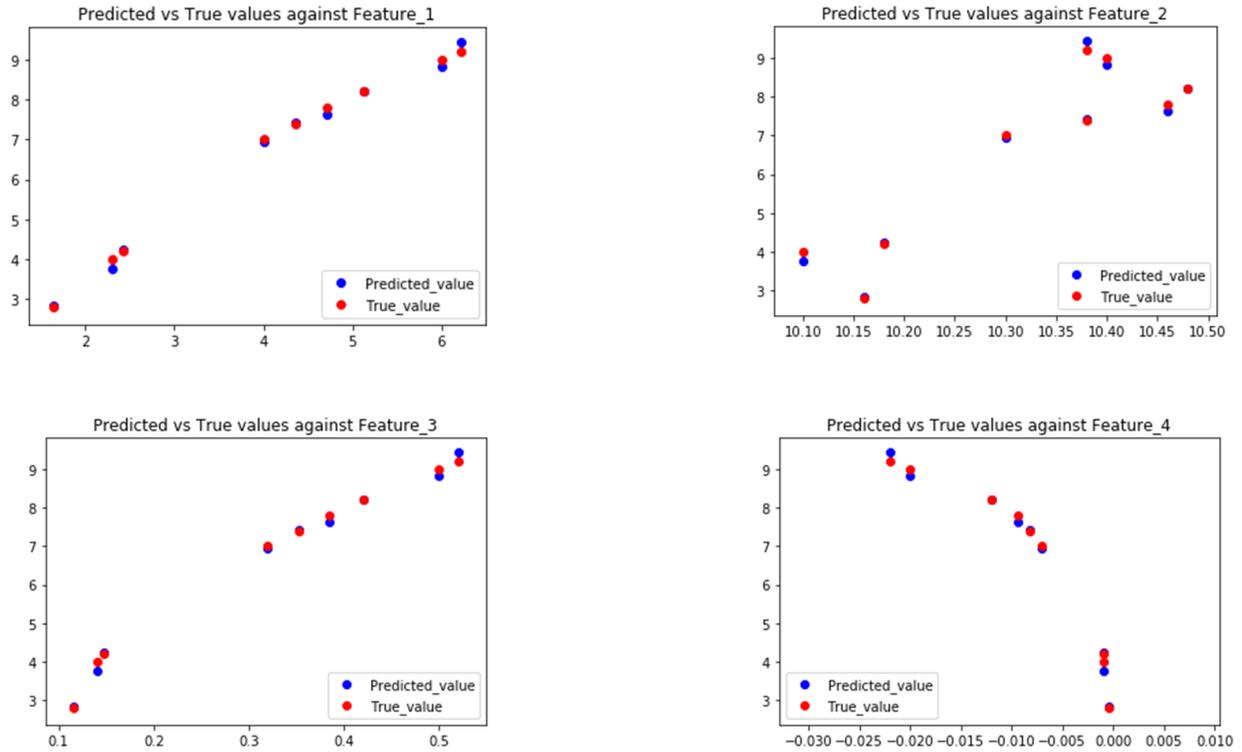


Figure 8: Predicted vs True values against different features

Next, we have implemented **Decision Tree** for regression. Here the criterion is the mean square error (mse) and the DecisionTreeRegressor model is imported from scikit learn. From the below figure it is clear that the random forest fits the data well than the decision tree regressor.

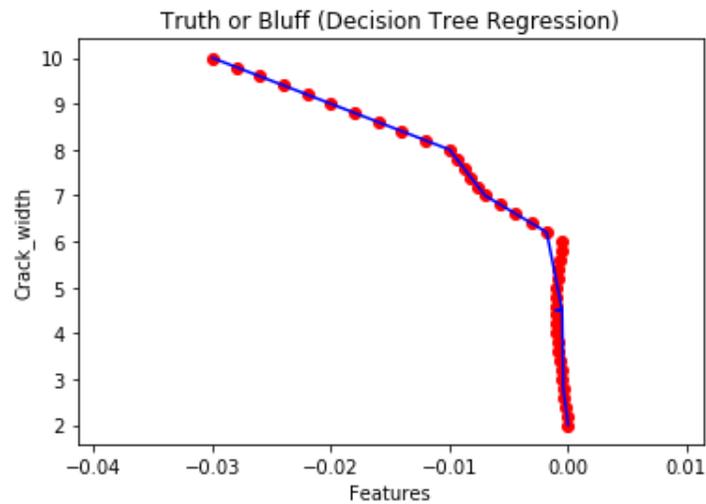


Figure 9: Decision Tree regression fitting the data

Next the table showing the true and predicted values are shown below.

Table 4: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
6.8	7
8.8	9
8	7.80
2.6	2.8
3.8	4
8	8.2
7.2	7.4
4.4	4.2
9.4	9.2

Next, we have plotted the predicted vs true values for different features using decision tree regression to show how accurate this algorithm is. From the below figures we can see that the predicted values are close to the true values not superimpose though, thus making the R score as **0.9918**.

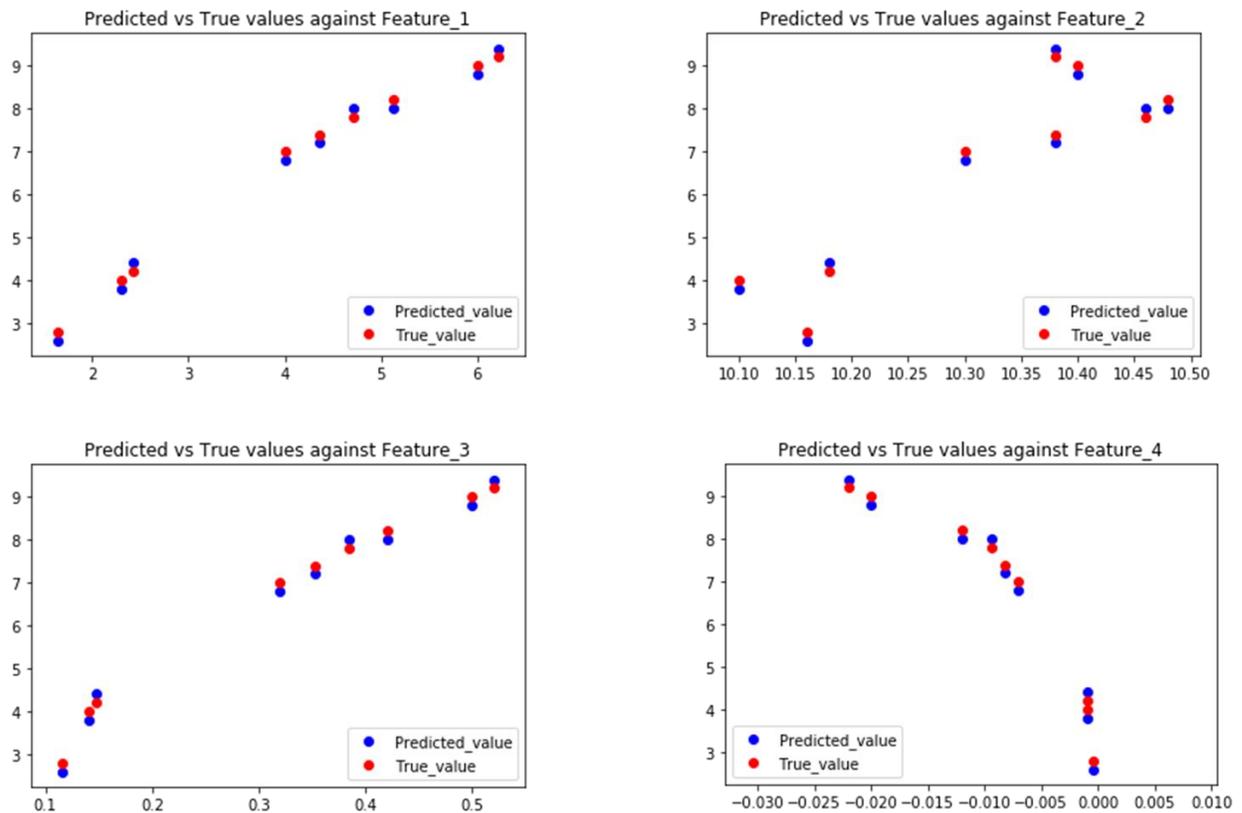


Figure 10: Predicted vs True values against different features

Next, we have implemented **Support Vector** for regression. Here as it is mandatory to perform feature scaling before proceeding with the regression hence, we have imported the

StandardScaler function to perform standardization. Thus, on the standardized data we have perform the regression. Here in SVR we have used radial basis function (rbf) as the kernel. The below plot shows the closeness of the predicted values against the true ones for a certain feature.

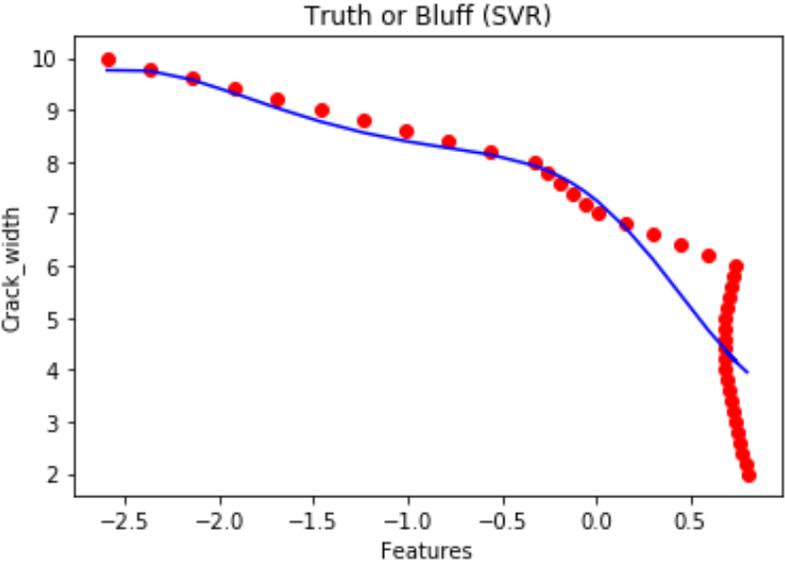


Figure 11: SVR regression fitting the data

Next the table showing the true and predicted values are shown below.

Table 5: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
7.1	7
8.85	9
7.72	7.80
2.87	2.8
3.9	4
8	8.2
7.49	7.4
3.88	4.2
9.08	9.2

Next, we have plotted the predicted vs true values for different features using SVR regression to show how accurate this algorithm is. From the below figures we can see that the predicted values are close to the true values not superimpose though, thus making the R score as **0.9950**.

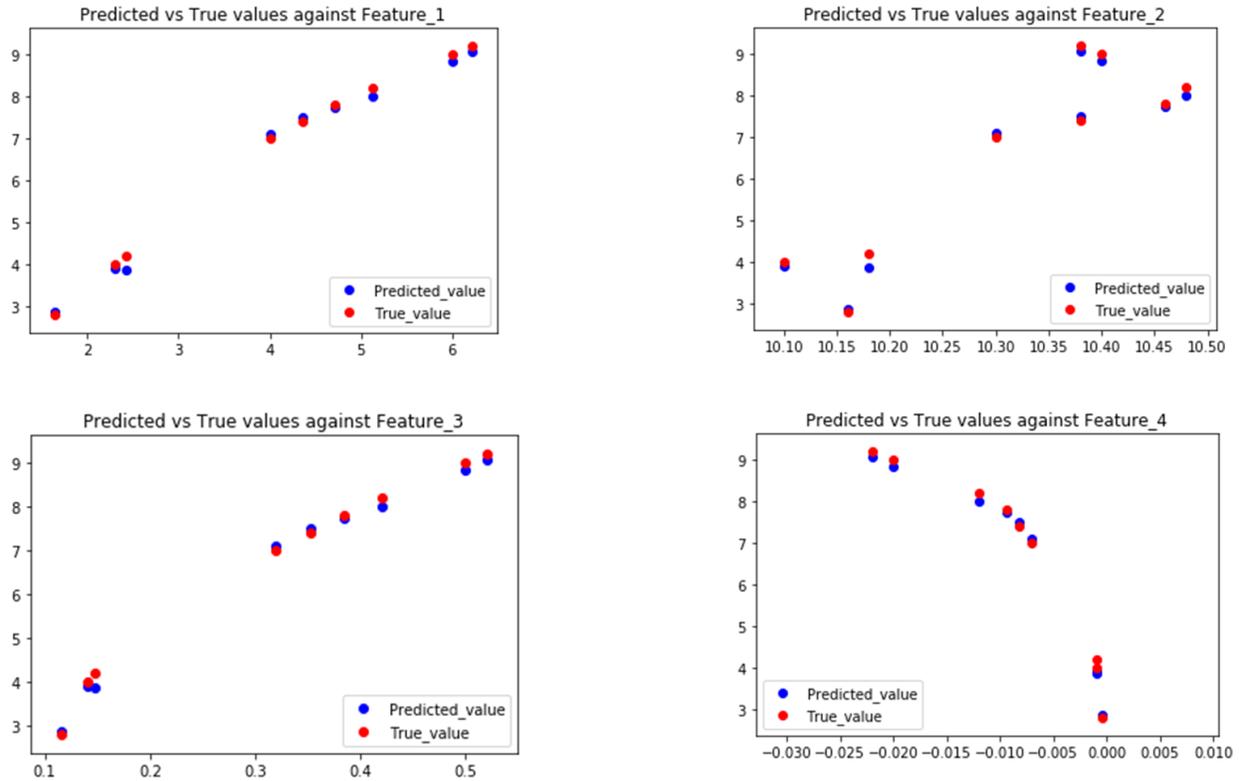


Figure 12: Predicted vs True values against different features

Next, we have implemented **Artificial Neural Network (ANN)** for regression. We have used TensorFlow for implementing the ANN. Like in the below figure our model consists of three simple layers with one input layer containing 4 features, then 2 hidden layers with each layer containing 6 neurons and finally the output layer containing the regressed crack width as output. The activation used in the hidden layers is relu.

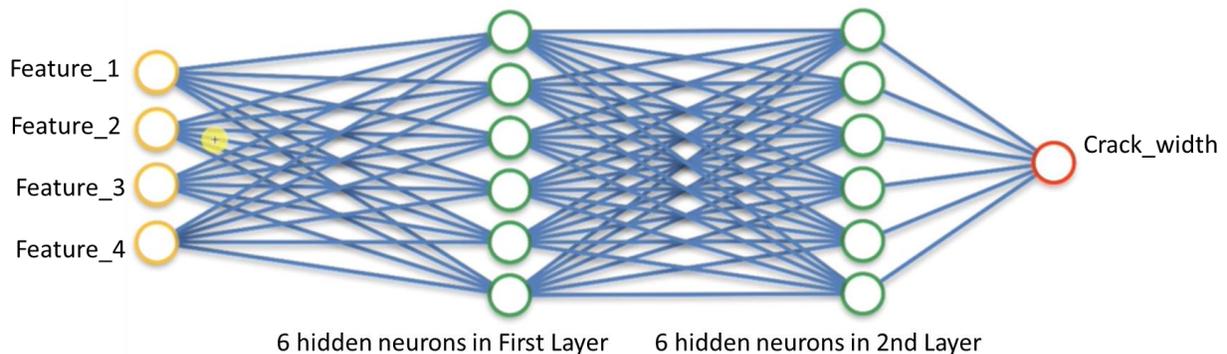


Figure 13: Model of the designed ANN

Now since it is a regression problem, hence for output we don't have to use any activation. Otherwise if it's a classification problem, then either we have to use softmax (for multi class) or sigmoid (for binary classification) activation functions. For back propagation we are using adam

optimizer with mean square error as the performance metric. We have run it for 4000 epochs with a batch size of 32 as our model is a simple one with each epoch taking only $31\mu s$. Below figure shows how the loss has converged with the passage of each epoch.

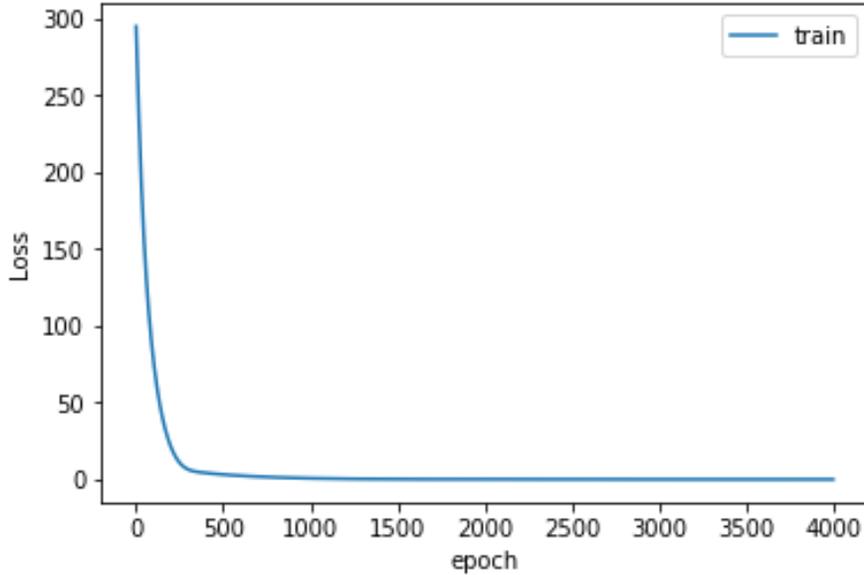


Figure 14: Loss vs epochs

Next the table shows the true and predicted values as obtained for the test set after training by the ANN.

Table 6: Predicted and true defect widths. (dimensions are in mm)

y_{pred}	y_{true}
7.18	7
8.99	9
8.07	7.80
2.81	2.8
4.02	4
8.37	8.2
7.78	7.4
4.23	4.2
9.15	9.2

Next, we have plotted the predicted vs true values for different features using ANN regression to show how accurate this algorithm is. From the below figures we can see that the predicted values are close to the true values not superimpose though, thus making the R score as **0.9936**.

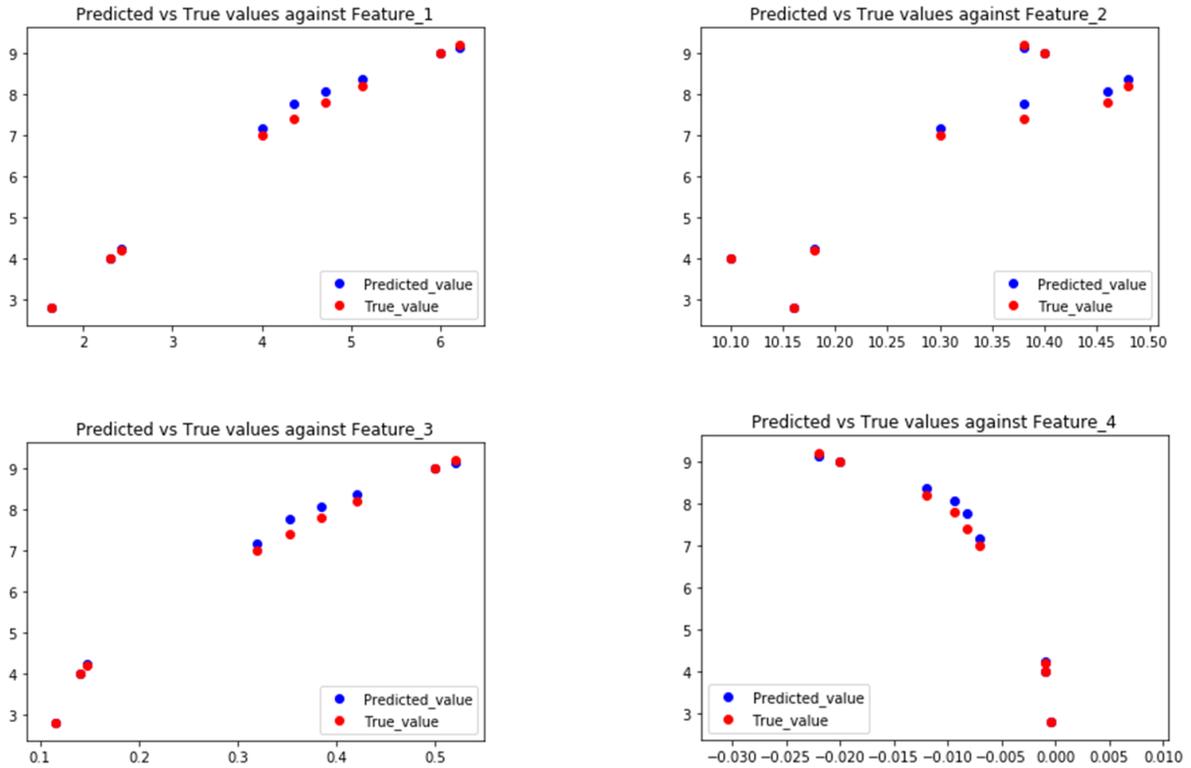
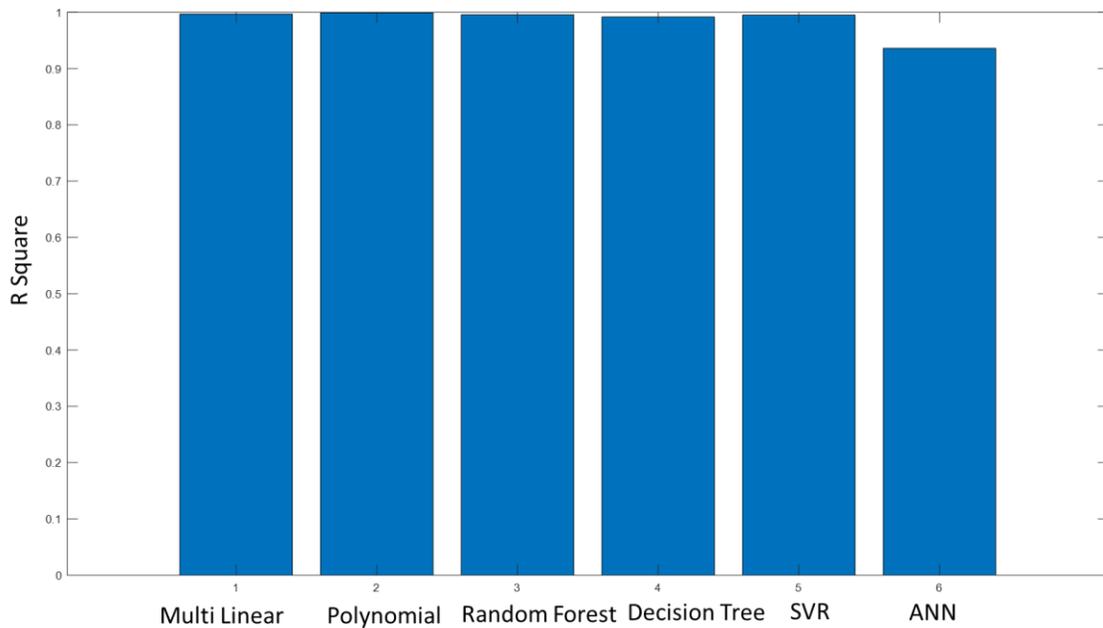


Figure 15: Predicted vs True values against different features



The above figure shows the bar plot of Rsquare against various regression models. Here the data in the training set was only 32 and in the test set was only 9. As ANN is a data hungry network

hence it needs lots of data. In future we will provide lots of data from the simulations obtained from COMSOL and feed into these models.

Classification into defect and non-defect classes: Here for classification we have 58 features and 200 samples. The train test split is done in a random way such that the train class contains 150 samples and test set 50 samples. Here 5 mm crack size is considered as the threshold by observing various literatures. Hence, we have to classify the defects here into two classes: ones above the 5 mm as defects and the rest as non-defects. Here for classification we have considered various datasets 1> the dataset not contaminated with noise. 2> dataset having SNR as 3> dataset containing SNR as i.e highly contaminated with noise. Classification is done by various machine learning algorithms Logistic Regression, SVM with linear and rbf kernel, Decision Tree, Random Forest and Naïve Bayes classifiers where we have applied those classifiers on the original dataset as well as the dataset obtained after obtaining PCA(Principal component analysis).

By applying the XGBoost gradient boosting library the important features are shown in below figure based on F score which shows the below three features carrying most information.

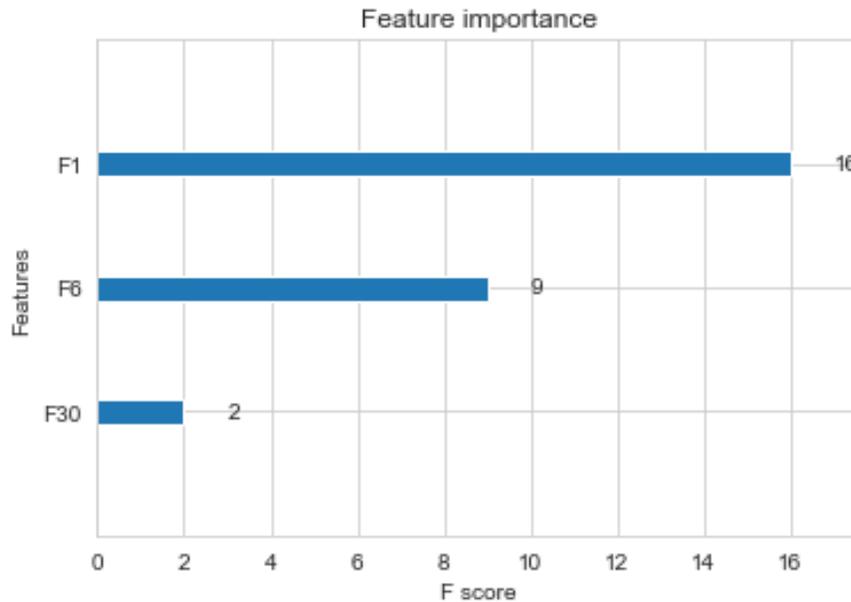


Figure 16: Features importance based on F score

Description of conducted analysis: As stated before here three type of datasets are considered to test the classification results. The PCA is used as the dimension reduction technique and thereby 5PC and 10PC directions are used as extracted features for two different datasets. The LDA is also used here as a dimension reduction technique. Different classifiers are then implemented on the dimension reduced data (5PCs and 10PCs) and the original data. In k-Nearest Neighbor, the number of neighbors is allowed to vary, and the optimal value of k is thereby chosen based on its performance on the test dataset. Then SVM, RF, Bayesian, decision tree, logistic regression classifiers are applied on the training dataset. The error rate and the performance metric are thereby evaluated on the test data set.

At first, we have used **Logistic regression** as the classifier. Only one out of 50 has been misclassified as the data is noiseless here. The error-rate is given as:

$$\text{error}_{\text{rate}} = \frac{\text{Number of test samples} - \sum \text{diagonal}(\text{confusion matrix})}{\text{Number of test samples}}$$

Hence by applying logistic regression the error-rate is **0.02** and the confusion matrix is obtained as:

Confusion matrix by LR=

	0 th label	1 st label
0 th label	23	0
1 st label	1	26

PCA is then implemented to reduce the dimension of data. The below figure shows the PCA based 2-dimensional dataset with red as (class 0) and dark green as (class 1).

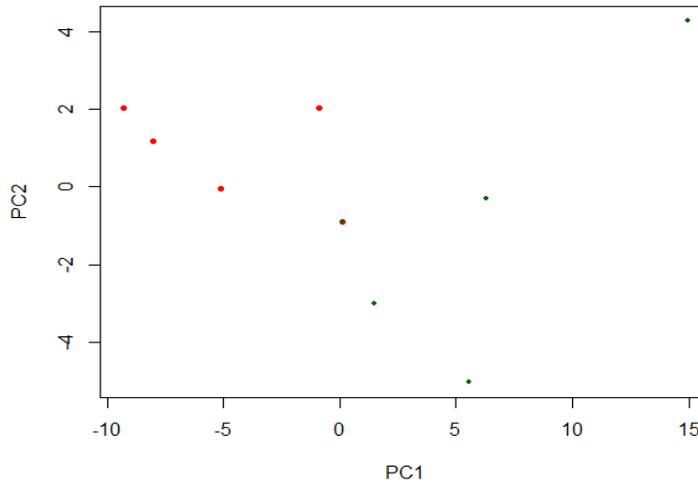


Figure 17: PCA based reduced 2-dimensional data set with colors red (label 1), and dark green (label 2)

Next in the bar plot, the contribution of different features in the first 2 major PC directions are shown. In the first PC direction, most features have a positive effect. But for the next PC direction, the positive and negative contributions look balanced.

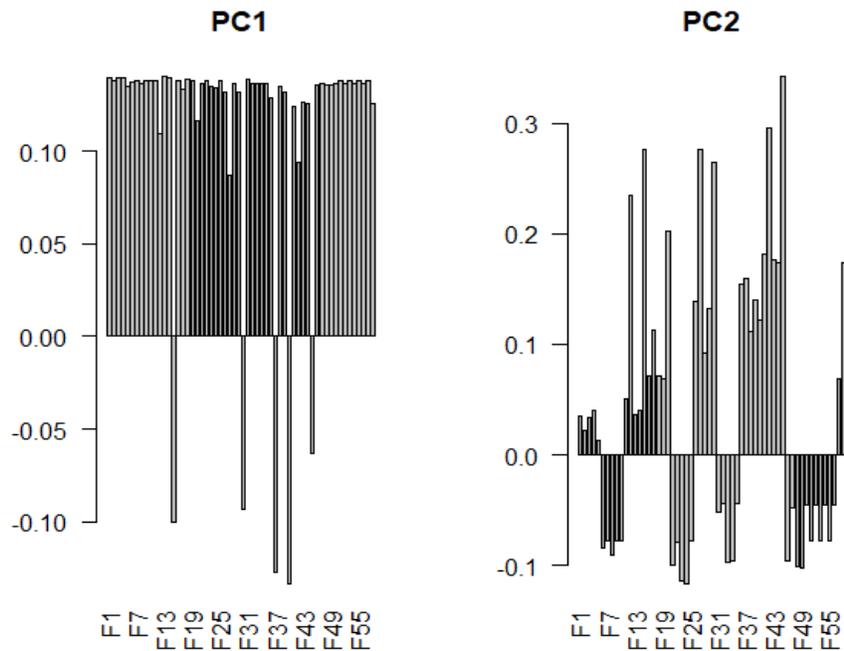


Figure 18: Bar plot showing the contributions of different features in the first 2 major PC directions.

Next we have applied **KNN (K-Nearest Neighbor) classifier** on both the original and reduced dataset.

The below figure shows the test error when the number of neighbors (k) is varied. From the below figure it is evident that changing value of k has no effect on the test error as it already minimized. On the original data set applying knn with number of neighbors as 1 gives the error rate as 0.02 whereas applying the same on the 5PC and 10 PC directions give error rate as 0.

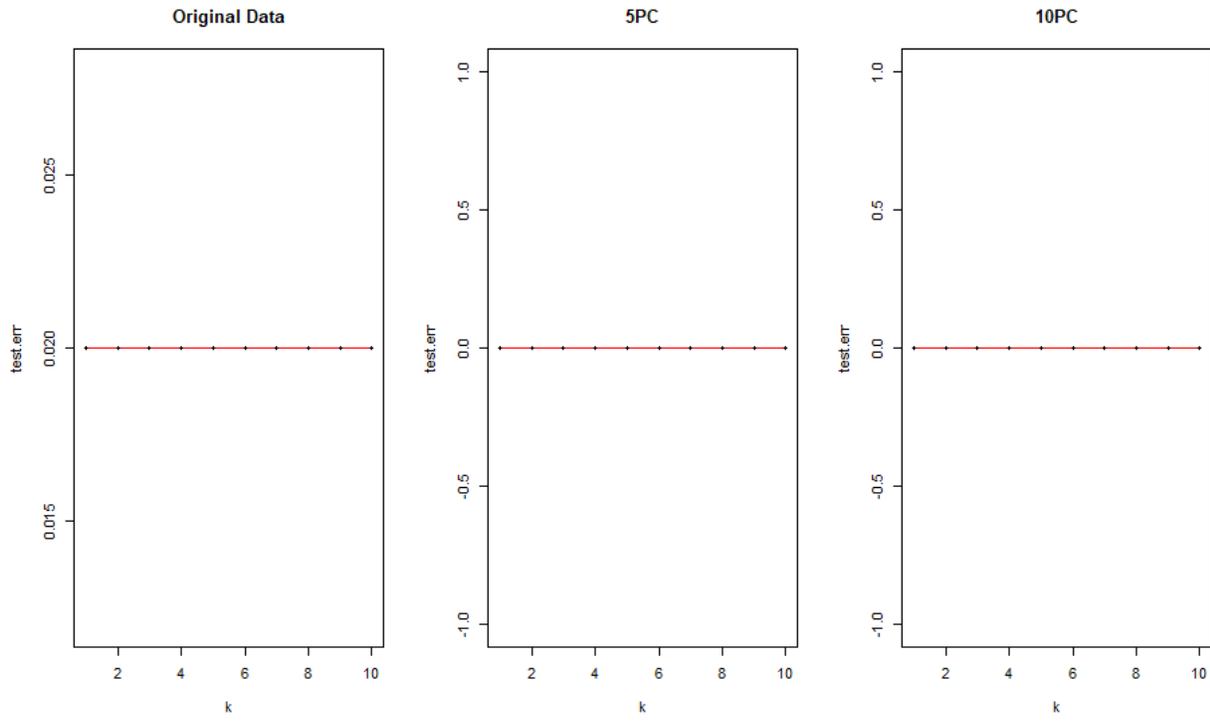


Figure 19: Test error vs the number of neighbors

Confusion matrix by KNN on original data set =

	0 th label	1 st label
0 th label	23	0
1 st label	1	26

Next, we have applied **SVM using both linear and rbf kernel** for classification. We have allowed the cost parameter of the SVM to vary between 0.5 to 5. However, the below figure shows that the training error and test error for all the cost parameters for this dataset is the least. Hence there is no effect of cost parameter in this dataset, as this dataset is already overfit due to the absence of noise.

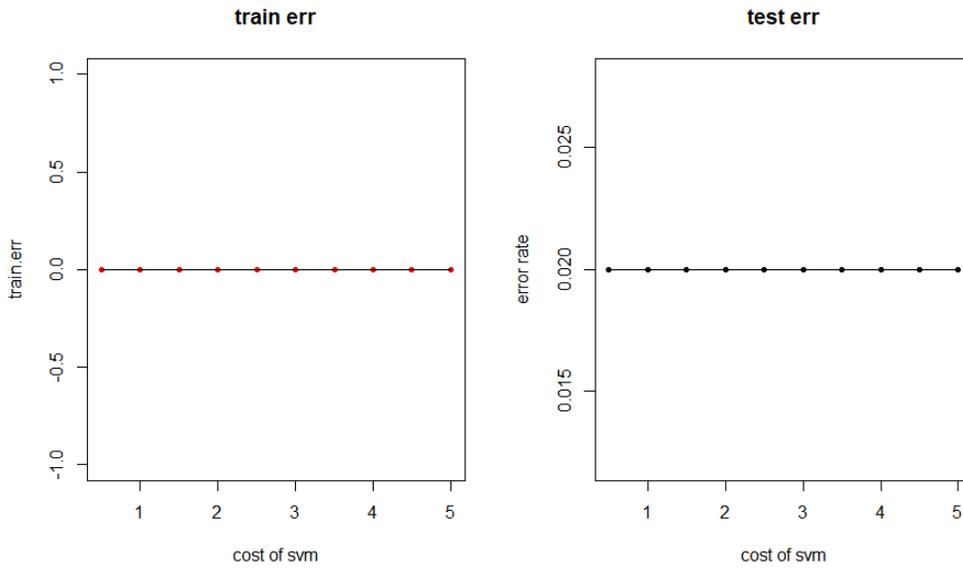


Figure 20: SVM against different cost functions.

The error rate here is 0.02 and the confusion matrix is same as given by:

	0 th label	1 st label
0 th label	23	0
1 st label	1	26

Next by applying **Decision Tree, Random Forest, Naïve Bayes** we achieved the error rate as 0 and the confusion matrix as:

	0 th label	1 st label
0 th label	23	0
1 st label	0	27

Next on the noisy dataset we have applied the classification. Here we have classified into three groups: crack size of width ranging from 1 to 3 mm as benign or class 0, crack size of width 4 to 6 mm as class 1 or can be detrimental, crack size above 6 mm as class 2 or need immediate attention.

As before at first, we have applied Logistic regression for classification into three classes. Here we obtained an error rate of **0.12**. The confusion matrix obtained is:

	0 th class	1 st class	2 nd class
0 th class	14	0	0
1 st class	6	8	0
2 nd class	0	0	22

The same error rate and confusion matrix is obtained by applying Naïve Bayes classification. PCA is then implemented to reduce the dimension of data. The below figure shows the PCA based 2-dimensional dataset with red as (class 0) and dark green as (class 1) and yellow as (class 2). From the below plot it is evident that there is substantial overlap between the classes particularly in lower dimensions. Hence, it is better to take a large number of principal component (PC) directions instead of 2 or 3 PCs.

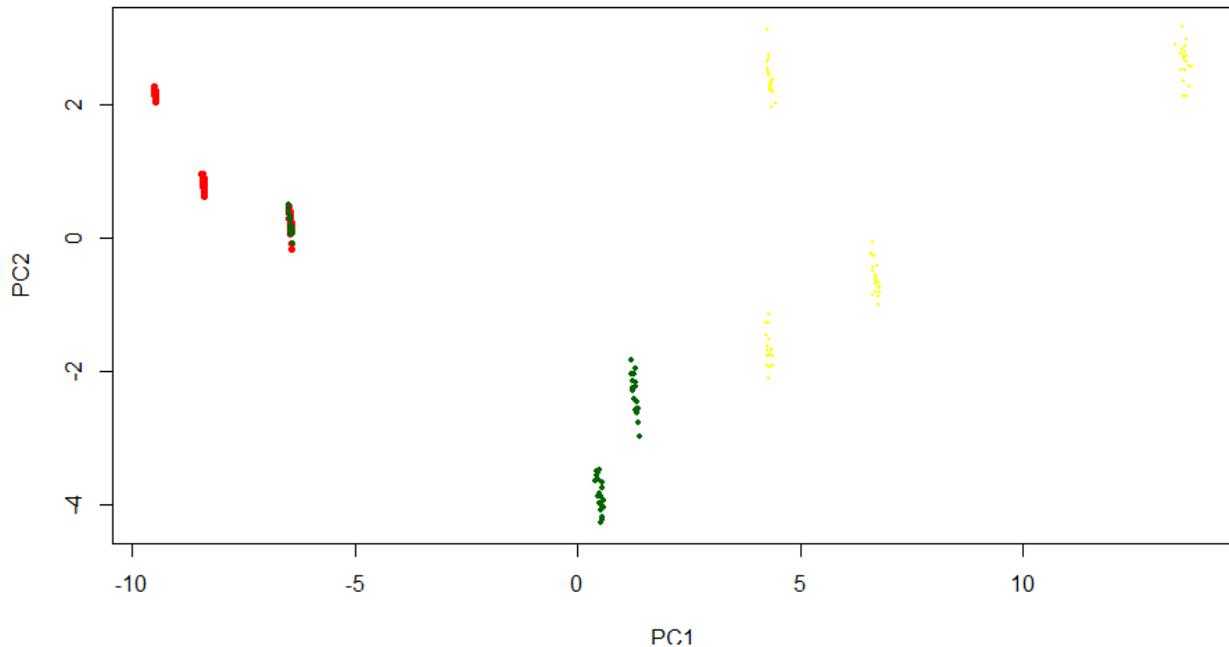


Figure 21: PCA based reduced 2-dimensional data set with colors red (label 1), dark green (label 2) and yellow (label3)

Next, we have applied **KNN (K-Nearest Neighbor) classifier** on both the original and reduced dataset.

The below figure shows the test error when the number of neighbors (k) is varied. Below figure shows applying KNN on original dataset gives a minimum test error of 0.02, whereas for the reduced datasets obtained by 5 PC and 10 PC the test error is always 0.

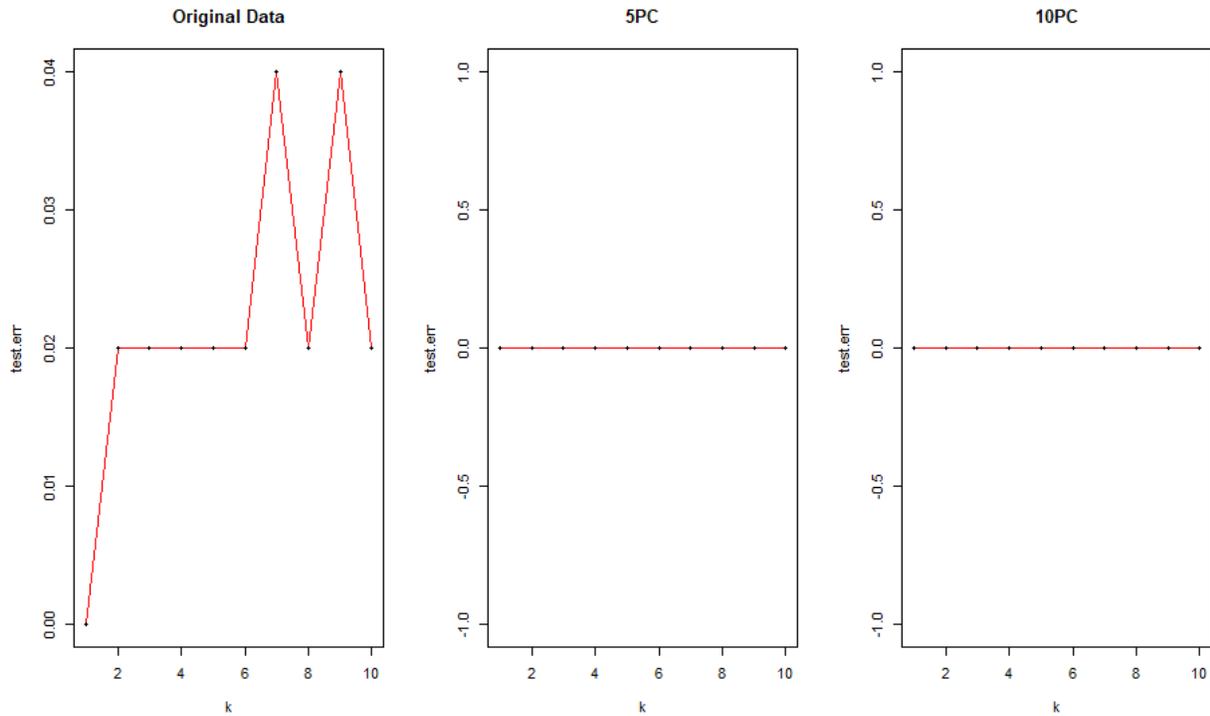


Figure 22: Test error vs the number of neighbors

Hence by applying KNN on the original dataset the error rate is **0.02** and the confusion matrix is:

	0 th class	1 st class	2 nd class
0 th class	14	0	0
1 st class	1	16	0
2 nd class	0	0	19

Next, we have applied **SVM using both linear and rbf kernel** for classification. We have allowed the cost parameter of the SVM to vary between 0.5 to 5. From the figure it is clear that at cost parameter at 2, the minimum classification error of **0.8** is obtained on the original dataset.

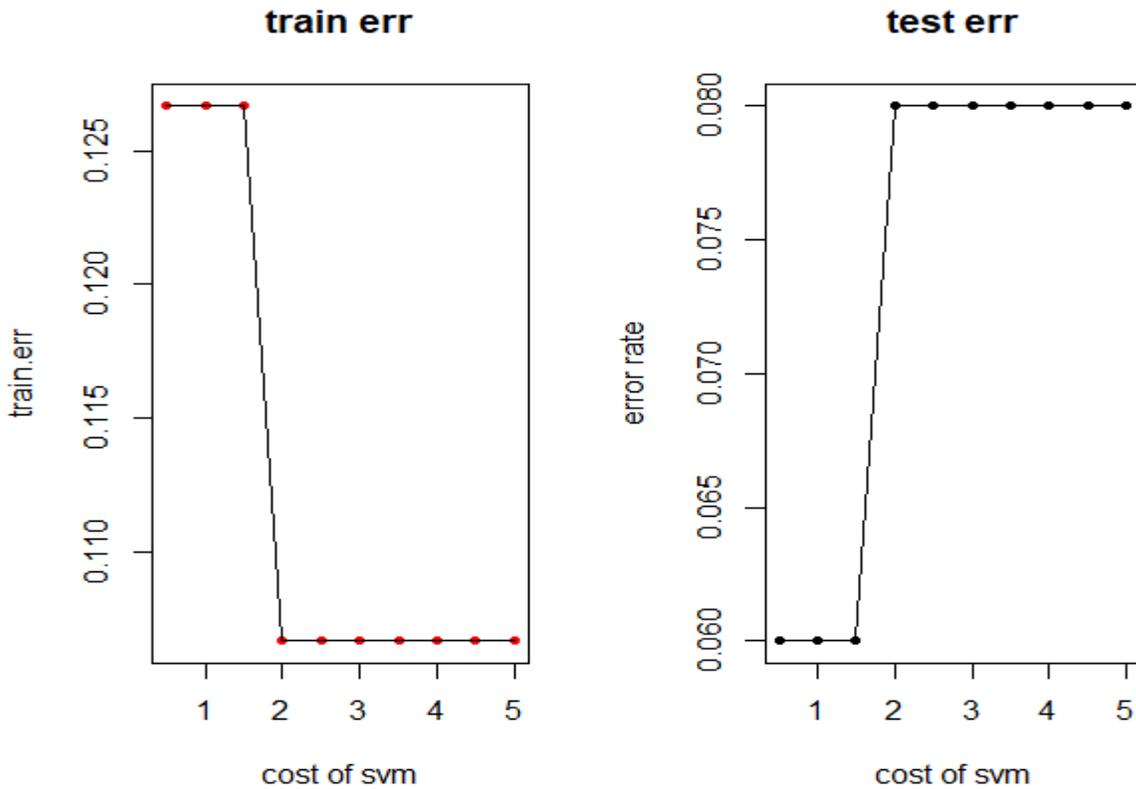


Figure 23: cost function in SVM

Thus, by applying SVM the error rate in classification is 0.08 with confusion matrix as:

	0 th class	1 st class	2 nd class
0 th class	10	0	0
1 st class	4	17	0
2 nd class	0	0	19

Similarly, by applying Random forest we have achieved an error rate of 0.8, whereas by applying decision tree the error rate increase to 0.12.

Finally, we have plotted all the error rates obtained from both the datasets by various classifiers. From the bar plot it is evident that with the increase in classes and noise in data the error rate increases.

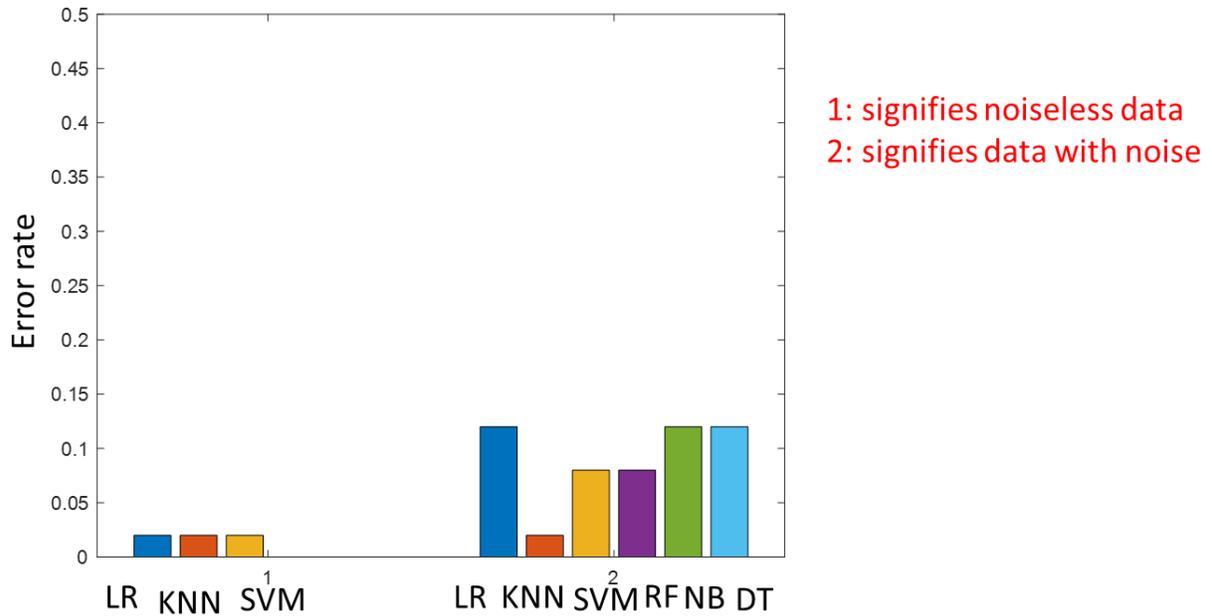


Figure 24: Bar plot showing error rate for various classifiers and various design of datasets

2. Progress on Task 4

2.1. Overview

This task focuses on developing a deep graph learning model for predictive interacting threat assessment. Predication is critical for pipeline operators to estimate threat evolution and the trend of pipeline integrity and perform necessary early intervention in order to prevent disastrous failures. Prediction typically requires the modeling and integration of a history of temporal data (e.g., conventionally, through trending). Recently, deep learning has shown significant potential for time series prediction. Especially, the notable success of deep spatial temporal graph neural network (ST-GNN) as proven their capability on many time-dependent predictions tasks. ST-GNN is a class of artificial neural network where connections between units form a directed graph along a sequence to encode dynamic temporal behavior for a time sequence. Although ST-GNN could be a great solution for predicting an individual pipe threat, they are unable to predict interacting threats, due to their lack of the capability to model the interactions between interacting threats

A key novelty in this project is a *deep graph learning* model that systematically embeds a spatiotemporal graph representation under ST-GNN framework to provide the new predictive ability to prognose interacting threats. Our approach adopts a graph to model the interaction between the interacting threats. To incorporate time and model the evolution of the interacting threats, a spatiotemporal graph is adopted, where the nodes of the graph represent the pipe threats, and the edges represent their spatiotemporal relationships. Then, based on the spatiotemporal structure of the graph, a ST-GNN is automatically constructed to perform

predictive assessment (e.g., to predict the Probability of Failure (POF) and Remaining Useful Life (RUL)).

Figure 25 shows our deep graph learning representation for interacting threat prediction, where an example of the interacting threat prediction problem is shown at the bottom, its spatiotemporal graph representation is shown at the top.

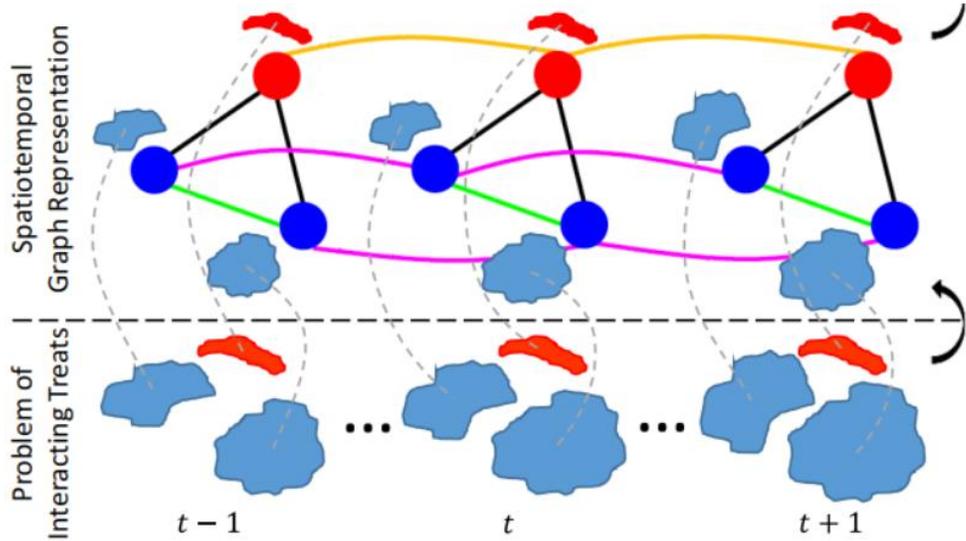


Figure 25: Our deep graph learning representation for interacting threat prediction

Given the spatial temporal graph representation of interacting threat prediction, we employ ST-GNN to do spatial temporal convolution with the representations, the ST-GNN model is shown in Figure 26. The overall process includes three steps, including spatial graph convolution, temporal graph convolution and the fusion of both spatial-temporal convolutions.

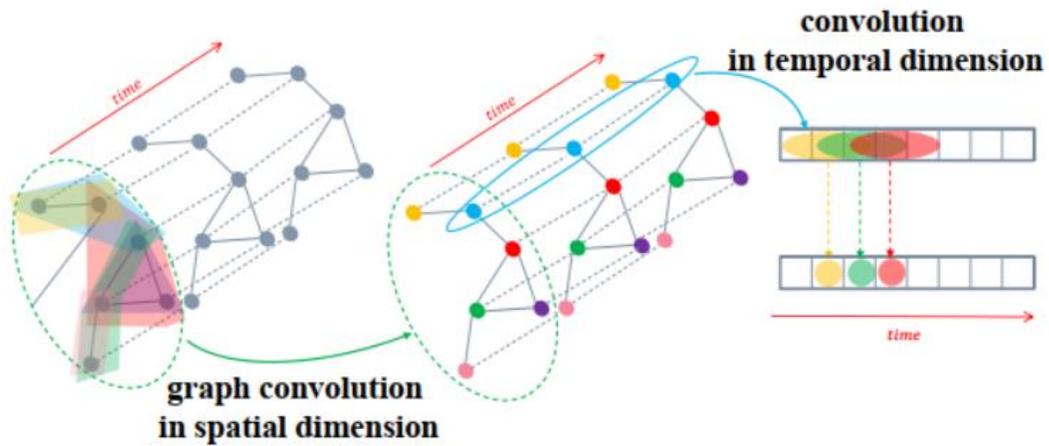


Figure 26: Spatial-temporal GNN convolution

2.2. Approach Details

2.2.1 Modeling Interacting Threats with Graphs

We model the interaction of threats at a time point as a spatial graph, where each node models a threat, and an edge connecting two nodes models the relationship of the pair of threats (e.g., their geometrical distance and/or relative orientation). Then, this spatial graph is unrolled in time to model the temporal evolution of these interacting threats, resulting in a spatiotemporal graph. The spatiotemporal graph can be denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}_S, \mathcal{E}_T)$, in which the spatial graph $(\mathcal{V}, \mathcal{E}_S)$ unrolls over time through edges \mathcal{E}_T . Figure 27 illustrates an example of using a spatiotemporal graph to model the interacting threats consisting of two corrosion threats and a crack. In the unrolled spatiotemporal graph, the nodes at a given time step t are connected with an undirected spatial edge e to model the threat interaction, and the nodes at adjacent time steps are connected with an undirected temporal edge to model threat evolution.

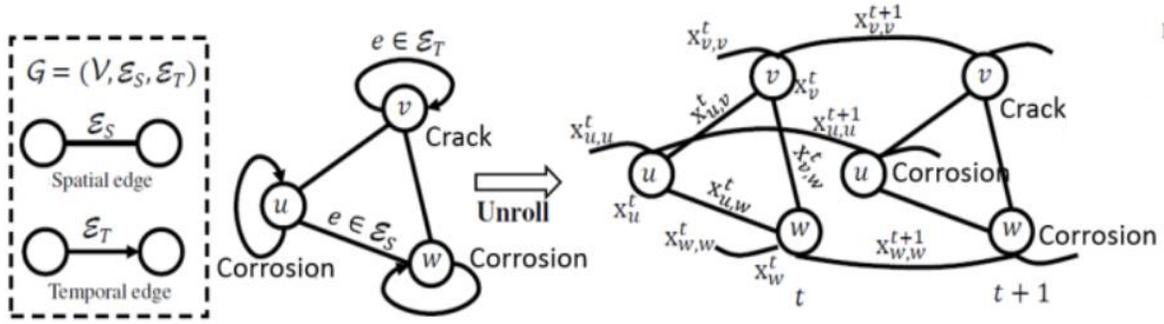


Figure 27: Spatiotemporal graph for interacting threat modeling

2.2.2 Problem Formulation

Given a spatiotemporal graph, with the feature vectors associated with the nodes x_v^t and edges x_e^t as the input, the goal is to predict values of the output nodes y_v^t at each time step t . In interacting threat prediction, the node features can be the learned representation vectors and/or threat characterization results (from Task 3), and the edge features can be the geometrical distance and/or relative orientation; the output nodes can be POF. The value of the output nodes y_v^t is determined by both the nodes and their interactions in all the history.

Formally, given $\mathbf{X}^t = \{\mathbf{X}_v^t, \mathbf{X}_e^t\}$ where $\mathbf{X}_v^t = [x_{v1}^t, x_{v2}^t, \dots, x_{vn}^t]$ and $\mathbf{X}_e^t = [x_{e1}^t, x_{e2}^t, \dots, x_{em}^t]$, where N is the number of nodes and m is the number of edges, t is the time step, we aim to predict POF or RUL of nodes represented by $\mathbf{Y}_v^t = [y_1^t, y_2^t, \dots, y_n^t]$. Our spatial temporal prediction model is formulated as $P(\mathbf{Y}^{t+k} | \mathbf{X}^t, \mathbf{X}^{t-1}, \dots, \mathbf{X}^1)$, where k denotes k time steps in the future.

In order to predict POF of interacting threats denoted as nodes, we design the spatial-temporal graph neural network as shown in Figure 28.

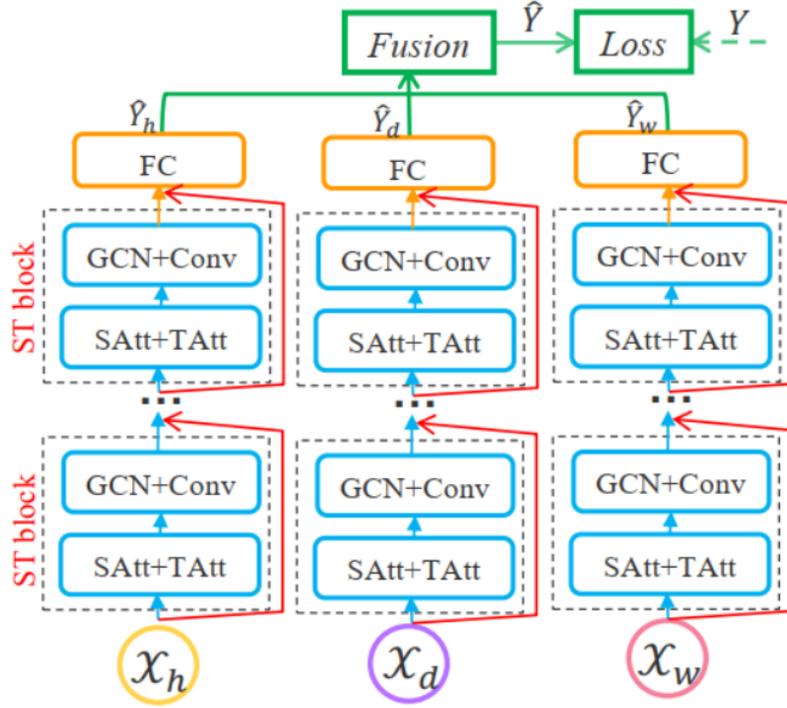


Figure 28: The architecture of the spatial-temporal graph neural network

Different time series: since different times of evolution of interacting threats may have different influence POF, we use three different time sequences X_h, X_d, X_w for the prediction, where $X_h = \{\mathbf{X}^{t-T_h}, \dots, \mathbf{X}^{t-1}, \mathbf{X}^t\}$ denoting a segment of one hour historical time series directly adjacent to the predicting time. Similarly, $X_d = \{\mathbf{X}^{t-T_d}, \dots, \mathbf{X}^{t-1}, \mathbf{X}^t\}$ and $X_w = \{\mathbf{X}^{t-T_w}, \dots, \mathbf{X}^{t-1}, \mathbf{X}^t\}$ denote one day and one week historical time series directly adjacent to the predicting time.

SAtt and Tatt: we also apply attention mechanism in our model to capture the dynamic spatial and temporal correlations on interacting threats. In spatial dimension, the interacting threats of different locations have influence among each other and the mutual influence is variant, we use an attention mechanism to adaptively capture the correlations between nodes in the spatial dimension. Formally, the spatial attention matrix \mathbf{S}_h for X_h is defined as follows:

$$\mathbf{S}_h = \mathbf{V} \sigma\left(\left(\mathbf{X}_h^{(r-1)} \mathbf{W}_1\right) \mathbf{W}_2 \left(\mathbf{X}_h^{(r-1)} \mathbf{W}_3\right)^T + \mathbf{b}_s\right)$$

where $\mathbf{X}_h^{(r-1)}$ denotes the r -th spatial-temporal block in the network (Figure 4, ST block), σ denotes the sigmoid activation function that is used to add nonlinearity into the model and $\mathbf{V}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3, \mathbf{b}_s$ are all learnable parameters. In order to normalize the attention matrix \mathbf{S}_h , we further apply SoftMax function to ensure each row in \mathbf{S}_h adding up to 1. Like \mathbf{S}_d and \mathbf{S}_w . The attention matrix \mathbf{S}_h is dynamically computed according to the current input of this layer. The value of an element $\mathbf{S}_{i,j}$ in \mathbf{S}_h semantically represents the correlation strength between node i and node j .

In temporal dimension, there exist correlations between the interacting threats in different time slices, and the correlations are also varying in different situations. Similar to the spatial dimension, we formulate the temporal attention matrix as follows:

$$\mathbf{E}_h = \mathbf{Z} \sigma\left(\left(\mathbf{X}_h^{(r-1)} \mathbf{U}_1\right) \mathbf{U}_2 \left(\mathbf{X}_h^{(r-1)} \mathbf{U}_3\right)^T + \mathbf{b}_e\right)$$

where $\mathbf{Z}, \mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3, \mathbf{b}_e$ are all learnable parameters.

GCN+Conv: the spatial-temporal attention module let the network automatically pay relatively more attention on valuable information. The input adjusted by the attention mechanism is fed into the spatial-temporal convolution module, whose structure is presented in figure 2. The spatial-temporal convolution module proposed here consists of a graph convolution in the spatial dimension, capturing spatial dependencies from neighborhood and a convolution along the temporal dimension, exploiting temporal dependencies from nearby times. In spatial convolution, we use spectral graph convolution to extract structural features of spatial graph, which is defined as follows:

$$g_\theta *_G x = g_\theta(\mathbf{L})x = g_\theta(\mathbf{U}\boldsymbol{\lambda}\mathbf{U}^T)x = \mathbf{U}g_\theta(\boldsymbol{\lambda})\mathbf{U}^T x$$

Where $*_G$ denotes a graph convolution operation, g_θ denotes a convolutional kernel function, \mathbf{L} is the Laplacian matrix that is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{U}\boldsymbol{\lambda}\mathbf{U}^T$ with degree matrix \mathbf{D} and adjacent matrix \mathbf{A} , a diagonal matrix $\boldsymbol{\lambda}$, and \mathbf{U} being the Fourier basis obtained from eigenvalue decomposition.

After the graph convolution operations having captured neighboring information for each node on the graph in the spatial dimension, a standard convolution layer in the temporal dimension is further stacked to update the signal of a node by merging the information at the neighboring time slice. Take the operation on the r th layer in the recent component as an example:

$$\mathbf{X}_h^{(r)} = \text{ReLU}(\Phi * (\text{ReLU}(g_\theta *_G x)))$$

where $*$ denotes a standard convolution operation, Φ is the parameters of the temporal dimension convolution kernel, and the activation function is ReLU.

FC: denotes fully connected layer. A fully connected layer is appended to make sure the output of each component has the same dimension and shape with the prediction target. The final fully connected layer uses ReLU as the activation function.

Fusion: the fusion stage is used to combine the outputs obtained from three sub-network for $\mathbf{X}_h, \mathbf{X}_d, \mathbf{X}_w$. The definition is as follows:

$$\hat{\mathbf{Y}} = \mathbf{W}_h \circ \mathbf{Y}_h + \mathbf{W}_d \circ \mathbf{Y}_d + \mathbf{W}_w \circ \mathbf{Y}_w$$

Where $\mathbf{W}_h, \mathbf{W}_d, \mathbf{W}_w$ are learnable parameters and \circ denotes element-wise production.

Loss: the loss function is defined as $\|\mathbf{Y} - \hat{\mathbf{Y}}\|_2^2$, where $\hat{\mathbf{Y}}$ is the prediction obtained from our model and \mathbf{Y} denotes the ground truth of POF.

By optimizing the above spatial-temporal graph neural network with the loss function, we could learn the parameters in a supervised fashion using data with ground truth. Then using the model to predict POF for interacting threats, with each element in $\hat{\mathbf{Y}} = [\hat{y}_1, \hat{y}_2, \dots, \hat{y}_n]$ denotes the failure pressure at the position of each threat.

3. Summary and Future Work

In this report and the previous report, we started working on Task 3.2 (Reliability analysis based on deep learning) and Task 4.2 (temporal model learning). In the next quarter, MSU will focus on developing deep learning methods for threat analysis and reliability analysis. Mines will implement interactive threat modeling based on spatiotemporal deep learning. Mines and MSU will also collaboratively wrap up the project in the next quarter.