Project 9: Structural Health Monitoring

This manuscript (permalink) was automatically generated from dbudzik/project9_SHM@0fa16c7 on December 5, 2020.

Authors

- **David Budzik**
  · dbudzik
- **Claudia Mederos**
  · cr25
- **Bolaji Lawal**
  · bolajilawal
- **Abdullah Assaf**
  · aboo12

Abstract

Civil infrastructure all around is subjected to the challenges posed by aging, deterioration and extreme events. In recent years, structural health monitoring has grown in importance as failure and collapse of infrastructure result in harmful effects on a nation's economy and development. Recent advances as seen the use of sensors to obtain the dynamic characteristics of structures. This has led to the potential of collecting dynamic data at more frequent intervals thus leading to continous monitoring. In this report, we use exploratory data analysis techniques to see if such dynamic data collected from structures can be used to infer the condition of a structure. We have also tried different machine learning algorithms to predict the condition of a structure based on given acceleration signals. For this study, the ASCE benchmark problem experimental phase II structure was used.

Machine Learning Methods

4. Overview

Following a comprehensive exploratory data analysis as outlined in the previous section, which was valuable in looking at different patterns and features of the data obtained from various sensors. The next task then, was to classify this data as either damaged or undamaged. Classification problems are very common in the data mining and machine learning applications. As such, several machine learning algorithms have been proposed and developed over the years for solving such problems. For our project, we have evaluated and tested five of those algorithms namely, Logistic regression, polynomial regression, artificial neural networks, recurrent neural networks and random forests. ## 4.1 Explanation of the Problem: Our Kaggle Project was focused on Binary classification. Binary Classification is the task of classifying the elements of a set into two groups on the basis of a classification rule. Our problem revolved around classifying the structural damage detected in a structure. We had to choose different machine learning algorithms to help classify the data.

4.2 Steps For each algorithm:

First we had to create a model, softmax or sigmoid used in output layer, so that we could train the model and tune hyperparameters, such as accuracy, precision etc. Finally, we had to test the performance of model against the sample data. This allowed us to compare results from different models and see what model was the most accurate at classifying the data.

4.3 Data Preparation:

First we had to split the train dataset into 2, one set for training and the other for validation. Then we had to shuffle the train dataset before using it. We then used regularizations to apply penalties on layer parameters or layer activity during optimization. Dropouts were used to incorporate non-linearity. The data was normalized by subtracting the mean and dividing by standard deviation.

4.4 Method 1: Logistic Regression:

Logistic regression is a technique commonly typically used for predicting binary classes but can also be used for multi-class problems and is adopted from the field of statistics. It describes and estimates

the relationship between one dependent variable and the independent variables. Logistic regression is a special case of linear regression that produces a constant output which is categorical in nature. Thus it is based on the linear regression equation:

$$y = \beta + w_1 x_1 + w_2 x_2 + w_3 x_3 ...$$

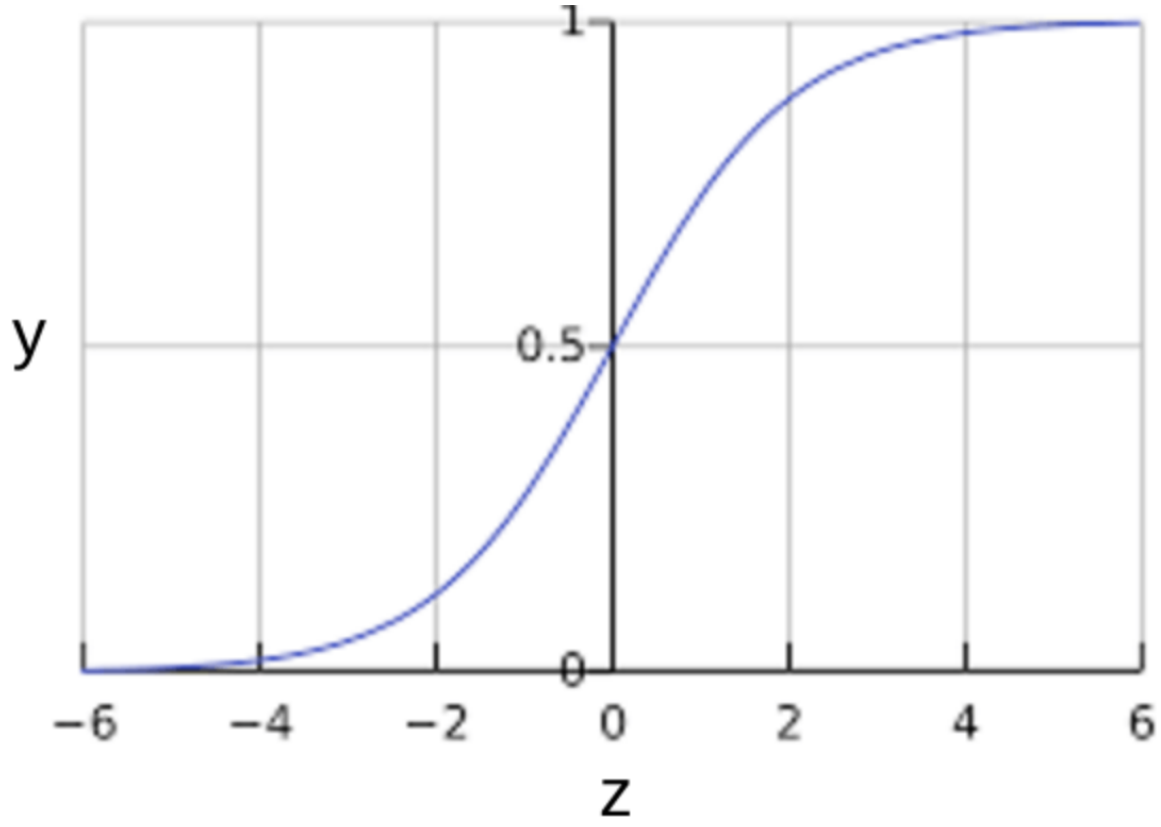

Figure 4.1: logistic_regression (sigmoid_activation_function)

4.4.1 Justification for Logistic Regression:

Logistic regression is one of the most simple machine learning algorithms that can be used for binary classification problems. It is easy to implement and can be used as a baseline for any binary classification problem. As such, we decided to use it as a baseline to which all the other models will be compared with. It can also be built upon for developing more complex machine learning algorithms for deep learning. It computes a probability ouptut and in order to map this output to a binary category we needed to define a classification threshold (also known as the decision threshold). However, a linear regression does not necessarily yield an output betwewn 0 and 1. Therefore,to ensure the output probability is always between 0 and 1 we used a sigmoid activation function in the output layer. Different combinations of hyperparameters were tested for this algorithm, however to save computation time an epoch value of 20 was used with a learning rate of 0.001. A classification threshold of 0.2 was also found to work best for the given dataset. Although logistic regression is sensitive to outliers and multicorrelation between the features, our exploratory data analysis showed we do not have this problem for our given dataset. Therefore, we expect to achieve reasonable results using this method.

```
  # Add the feature layer (the list of features and how they are represented)
  # to the model.
  model.add(feature_layer)

  # Funnel the regression value through a sigmoid function.
  model.add(tf.keras.layers.Dense(units=1, input_shape=(1,),
                                  activation=tf.sigmoid),)

  # Call the compile method to construct the layers into a model that
  # TensorFlow can execute.  Notice that we're using a different loss
  # function for classification than for regression.
  model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=my_learning_rate),
                loss=tf.keras.losses.BinaryCrossentropy(),
                metrics=my_metrics)
```

Figure 4.2: overview_of_logistic_regression_model

4.5 Method 2: Polynomial Regression: One common pattern within machine learning is to use linear models trained on nonlinear functions of the data. This approach maintains the generally fast performance of linear methods, while allowing them to fit a much wider range of data.

For example, a simple linear regression can be extended by constructing polynomial features from the coefficients. In the standard linear regression case, the model is based on the expression previously shown above. If we want to fit a paraboloid to the data instead of a plane, we can combine the features in second-order polynomials, so that the model looks like this:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 {x_1}^2 + w_5 {x_2}^2$$

The (sometimes surprising) observation is that this is still a linear model: to see this, imagine creating a new variable

$$z = [x_1,\ x_2,\ x_1 x_2,\ {x_1}^2,\ {x_2}^2]$$

With this re-labeling of the data, our problem can be written as:

$$y = w_0 + w_1 z_1 + w_2 z_2 + w_3 z_3 + w_4 z_4 + w_5 z_5$$

4.5.1 Justification for Polynomial Regression: Sometimes the relationship between the dependent and independent variables maybe non-linear. As in this case, we do not expect a linear relationship between the features and the target variable, it made sense to try a non-linear algorithm. The simplest and most common non-linear method to use is the polynomial regression as described earlier. Intuitively, we would expect a regression model with higher orders to perform better than a simple one. However, we didn't find significant improvements between 2nd degree and higher order polynomials. Therefore, for this project we chose to use 2nd degree polynomial regression models to help save computation time.

```
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree=2)

X_train_2_d = poly_reg.fit_transform(X_train)
X_test_2_d = poly_reg.transform(X_test)

lin_reg = LinearRegression(normalize=True)
lin_reg.fit(X_train_2_d,y_train)

test_pred = lin_reg.predict(X_test_2_d)
train_pred = lin_reg.predict(X_train_2_d)

print('Test set evaluation:\n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\n_____________________________________
_')
print_evaluate(y_train, train_pred)
```

Figure 4.3: overview_of_polynomial_regression_model

4.6 Method 3: Artificial Neural Networks:

The second method utilized was Artificial Neural Networks. Artificial Neural Networks are more complex than logistic regression. This method adds a bunch of hidden non-linear layers to the logistic regression model. Our group used this method to check if it offered an improvement on the previous model. The artificial neural networks yeiled average results with great accuracy ~96%, Improved precision ~45%, Improved recall ~60%, although the metrics were not good overall.

4.7 Method 4: Recurrent Neural Networks:

Recurrent neural networks (RNNs) are a very powerful tool, however they suffer from the vanishing gradient problem. In order to avoid this issue, we have used a better variation of RNNs called Long Short Term Networks (LSTM) for this project. This basically consists of cells that are responsible for “remembering” values over a time interval. This differs from the traditional neural netwok in that not only does it learn from the features, but it also takes care of sequence values over time. That is, for traditional neural networks it is assumed that all inputs and outputs are independent of each other. However, for RNNs the output depends on the previous computation. As a result of this, RNNs are very popular for sequential data such as time series because they perform much deeper understanding of sequence when compared with other algorithms. They are also applicable to any data that can be rearranged to resemble sequential data. In terms of visualization, one can think of RNN models as shown below:

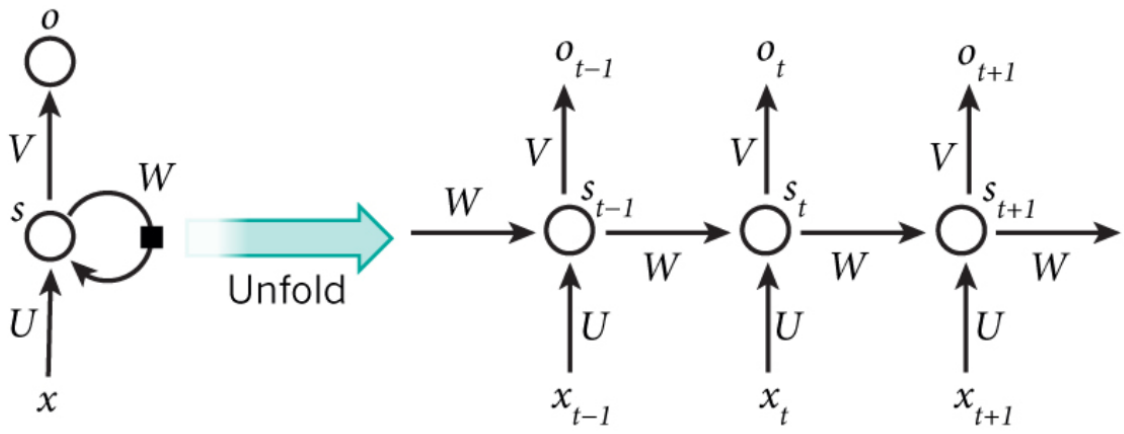

Figure 4.4: visualization_of_recurrent_neural_network_models

4.7.1 Justification for recurrent neural network:

Since, the index for our dataset actually represents time, then perhaps we can consider it as a time series data. As such, we decided to see how RNNs will perform in comparison with the other models. As was mentioned earlier, we used LSTM for this project. This type of model requires a 3-dimensional input, therefore we had to transform the shape of the dataset we had to make use of this model. In order to save computation time, a LSTM of 40 units was found to give the best performance and therefore used for testing the model. Additionally, a sigmoid activation function was applied in the outer layer. The hyperparameters were the same as with other models to ensure a fair coomparison between the different algorithms used.

```
model3 = tf.keras.Sequential()



model3.add(tf.keras.layers.LSTM(40))

model3.add(tf.keras.layers.Dense(units=1,activation='sigmoid'))
model3.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics= [
      tf.keras.metrics.BinaryAccuracy(name='accuracy',
                                      threshold=classification_thres
hold),
      tf.keras.metrics.Precision(thresholds=classification_threshol
d,
                                 name='precision'
                                 )
```

Figure 4.5: overview_of_recurrent_neural_network_model

4.8 Method 5: Random Forest Regression:

The third method utilized was Random Forest Regression. Random Forest Regression performs both regression and classification tasks with the use of multiple decision trees and bagging. It is easy to use and often returns good results even without hyperparameter tuning. Our group used this method to check if it offered an improvement on the previous model. The Random Forest Regression provided the worst results with extremely inaccurate accuracy rate, while also not working well for the type of data we had.

4.9 Issue:

We noticed that the problem was the precision and recall are very low even though accuracy is high. This was caused by the damaged data being too sparse, only 3.5% of our data represents the damaged condition. The solution was too add copies of the damaged data into the training set.

4.10 Take Two:

In take two we ran logistic regression and artifical nueral networks again to see if it yeilded better results with addiiton of new copies of damaged data. Logistic Regression returned good accuracy ~83%, much better precision ~60%, and Fantastic recall ~99%. Artifical Neural Network returned slighly better data with a fantastic accuracy ~98%, fantastic precision ~91%, and fantastic recall ~99%.

Results

5.1 Metrics

Metrics are an important aspect of machine learning because they provide an insight to the performance of the model. Therefore, it is important to think about the metrics that need to be used in order to properly judge a model's behavior and also allow the model to better improve its predictions. The metrics that were used in this project include root mean squared error, R^2 value, accuracy, precision, and recall.

5.1.1 Root Mean Square Error

Root mean square error is a metric commonly used in statistics for measuring the difference between predicted and actual values for a set of data. It is calculated by taking the square root of the sum of the squares of the difference between the predicted and actual values. Because of the way it is calculated, the RMSE can never be negative. A lower RMSE is better, meaning that the predictions are closer to the actual values.

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

Figure 5.1: Formula for RMSE [1]

5.1.2 R^2 Value

The R^2 value, also called the coefficient of determination, is a measure of how well a model is fit to the data. It gives an indication of how likely the model is to be able to predict a future value well based on the current data and outputs a number between 0 and 1, where higher numbers are desired. The R^2 calculation is based on the correlations between the independent variables and the correlations between the independent variables and the dependent variable. The equation can be found below in Figure 5.2. There, c is the vector containing the correlations between the independent and the dependent variables, and R is the correlation matrix of the dependent variables.

$$R^2 = \mathbf{c}^\top R_{xx}^{-1}\,\mathbf{c},$$

Figure 5.2: Formula for R^2 [2]

5.1.3 Accuracy

Accuracy, as it is used in machine learning, is a measure of the fraction of datapoints correctly identified by the model over the total amount of datapoints. It is calculated by the formula seen in Figure 5.3. It is one of the most common metrics for determining the results of a statistical problem.

$$\text{Accuracy} = \frac{\text{True Positives + True Negatives}}{\text{All Samples}}$$

Figure 5.3: Formula for Accuracy [3]

5.1.4 Precision

Precision is a measure of the fraction of datapoints identified correctly over the total amount of datapoints identified for a given problem. This shows how often datapoints are incorrectly identified by the model. It is calculated as given in Figure 5.4.

5.1.5 Recall

Recall is the ratio of how many identified datapoints make up the total set of that category of data. Recall is the indicator of how well the model is finding all of the data being sought. It is calculated as given in Figure 5.4.

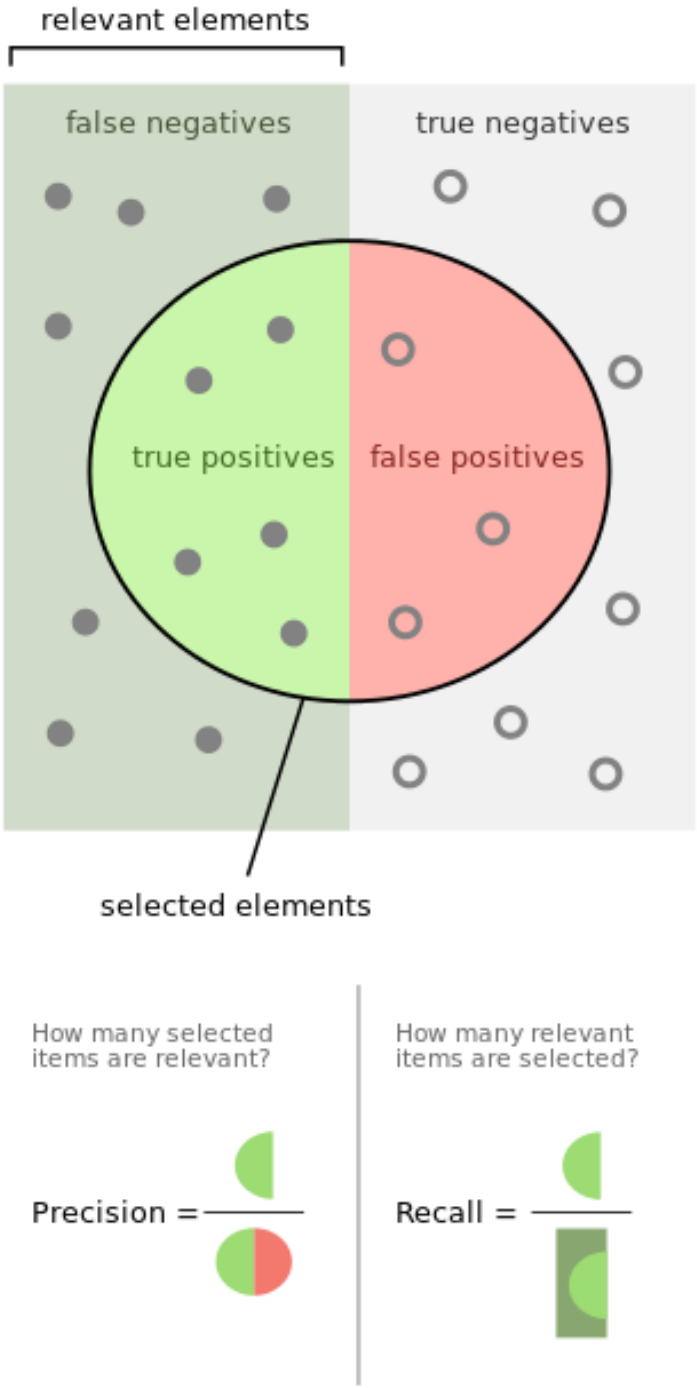

Figure 5.4: Formulas for Precision and Recall [4]

5.2 Logistic Regression

Logistic regression proved to perform relatively well in this experiment, especially once the dataset was augmented with additional copies of the sparse data. First looking at the results before the dataset was changed, the model showed some unexpected results, as seen in Figure 5.5. The results of the validation set matched closely to the training set, however this did not mean that the model was working properly. The accuracy was very high, around 96%, while the precision and recall ended up very low, both around 20%. This was a symptom of the sparsity of the dataset, as mentioned previously in the report. In addition, the recall started out very high and plummeted to a low value, which is a very unexpected behavior for the model. The model was clearly unable to learn the difference between an undamaged vs damaged data point. This is evidence of the fact that precision and recall are much more important metrics than accuracy for our experiment with the given dataset.

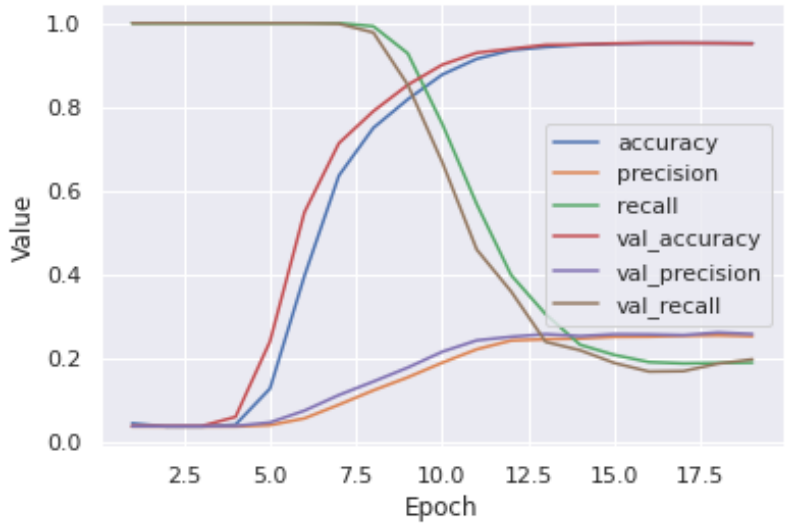

Figure 5.5: Results of Logistic Regression Model Before Dataset Augmentation

Once the dataset was augmented, the model produced a much improved result. As seen in Figure 5.6, the recall and precision are much improved from before. The recall was now 99%, and the precision went up to ~60%. The accuracy did suffer, now ~83%, however the model is much improved overall. The results showed that this model was correctly identifying almost all damaged datapoints, given the nearly perfect recall, and was incorrectly identifying a good amount of undamaged datapoints as damaged. This is not an ideal result but it is much better than missing damaged datapoints.

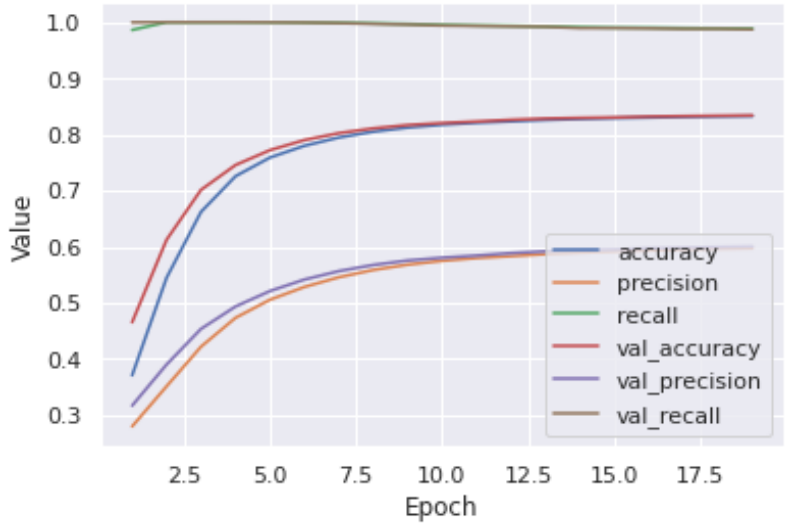

Figure 5.6: Results of Logistic Regression Model After Dataset Augmentation

5.3 Polynomial Regression

Polynomial regression did not perform very well in this experiment. As seen in Figure 5.7, the low R^2 value of ~0.175 in the validation set showed that the predictions of the model were not very well correlated with the actual values. Once the dataset was augmented and rerun through the model, the R^2 value greatly increased to around 0.537. This result is shown in Figure 5.8. Although this result is still quite low, the improvement is very noticeable and shows that a more refined model using polynomial regression may be useful. The RMSE of each run can also be compared. The first run had a lower score of 0.167 in comparison to the 0.292 of the second run. This is not what was expected, since a better model should have both a lower RMSE and a higher R^2 value. This result may be explained by the predictions of the model in the second run being more well correlated but having a

larger variance from the actual values. This could likely have been due to the fact that a much larger fraction of the data after augmentation was the damaged condition, which increased the variance of the dataset.

```
Test set evaluation:
----------------------------------
RMSE: 0.16681445337668216
R2 Square 0.17549483728169013
==================================
Train set evaluation:
----------------------------------
RMSE: 0.16728229185764404
R2 Square 0.1770909968453942
```

Figure 5.7: Results of Polynomial Regression Model Before Dataset Augmentation

```
Test set evaluation:
----------------------------------
RMSE: 0.29228008649886505
R2 Square 0.5374867907568724
==================================
Train set evaluation:
----------------------------------
RMSE: 0.29307223237197116
R2 Square 0.5395556169239921
```

Figure 5.8: Results of Polynomial Regression Model After Dataset Augmentation

5.4 Artificial Neural Network

The artificial neural network performed the best out of all the methods attempted. Using the original data, this model had a similar result to the logistic regression model, although with improved precision and recall of ~45% and ~60%, respectively. These results can be seen in Figure 5.9.

Figure 5.9: Results of Artificial Neural Network Model Before Dataset Augmentation

After the dataset was augmented and the model re-ran it produced a fantastic result that was far better than the rest of the models that were used. The model now had an accuracy of about 98%, a precision of ~91%, and a nearly perfect recall of 99.8%. This can be seen in Figure 5.10. These results show that the issue of sparse data was the only issue preventing this model from performing very well. It should be noted that the validation set precision did deviate away from the training set precision, which was likely a limitation of how well the model was able to learn the particularities of the dataset, which resulted in some undamaged datapoints being falsely labeled as damaged. Nevertheless, the extremely high recall shows that almost all of the damaged datapoints were found correctly, which was far more important.

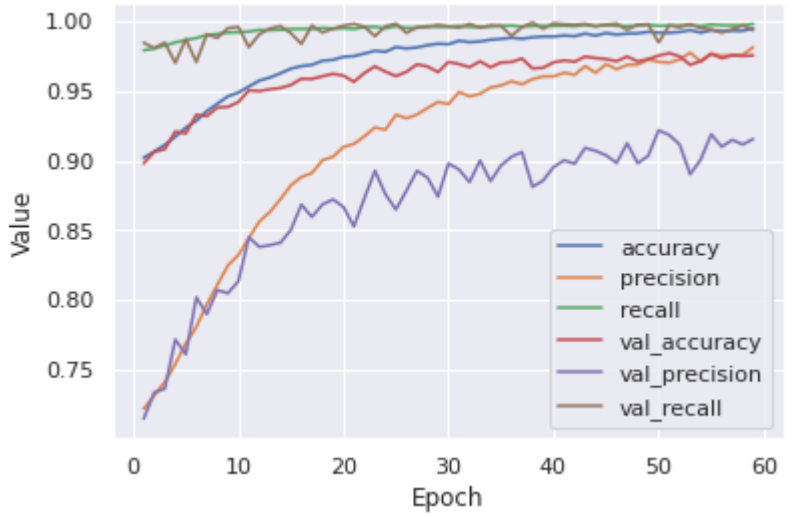

Figure 5.10: Results of Artificial Neural Network Model After Dataset Augmentation

5.5 Random Forest Regression

The random forest regression model can be most closely compared to the polynomial regression model. Before the dataset was augmented, this modal fared worse than the polynomial regression with an R^2 value of 0.092. The results are shown in Figure 5.11. It is seen that this model is a very poor fit for the type of dataset used in this project, since it did the worst out of the 4 models tried. After dataset augmentation, this model also vastly improved, up to an R^2 of 0.408, but was still not satisfactory. These results can be found in Figure 5.12. This model also shared the peculiar result of a higher RMSE after the augmented dataset was run and can be similarly explained.

```
Test set evaluation:

-----------------------------------
RMSE: 0.17506760173939853
R2 Square 0.09189181374302258
Train set evaluation:

-----------------------------------
RMSE: 0.17555483713913608
R2 Square 0.0936885337737473
```

Figure 5.11: Results of Random Forest Regression Model Before Dataset Augmentation

```
Test set evaluation:
--------------------------------------
RMSE: 0.3307290084792896
R2 Square 0.4077974625510703
Train set evaluation:
--------------------------------------
RMSE: 0.3318000408441319
R2 Square 0.4098251410867356
```

Figure 5.12: Results of Random Forest Regression Model After Dataset Augmentation

Discussion of Results

Discussion goes here

Conclusion

Conclusion goes here

References

1. **What does RMSE really mean?**
   James Moody
   Medium (2019-09-06) https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e

2. https://wikimedia.org/api/rest_v1/media/math/render/svg/2543e0a46d792a8bb93c4789e5aa10a7a3693cbc

3. https://lawtomated.com/wp-content/uploads/2019/10/Accuracy_2.png

4. https://upload.wikimedia.org/wikipedia/commons/thumb/2/26/Precisionrecall.svg/350px-Precisionrecall.svg.png