Project 9: Structural Health Monitoring

This manuscript (permalink) was automatically generated from dbudzik/project9_SHM@f9252b6 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:

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

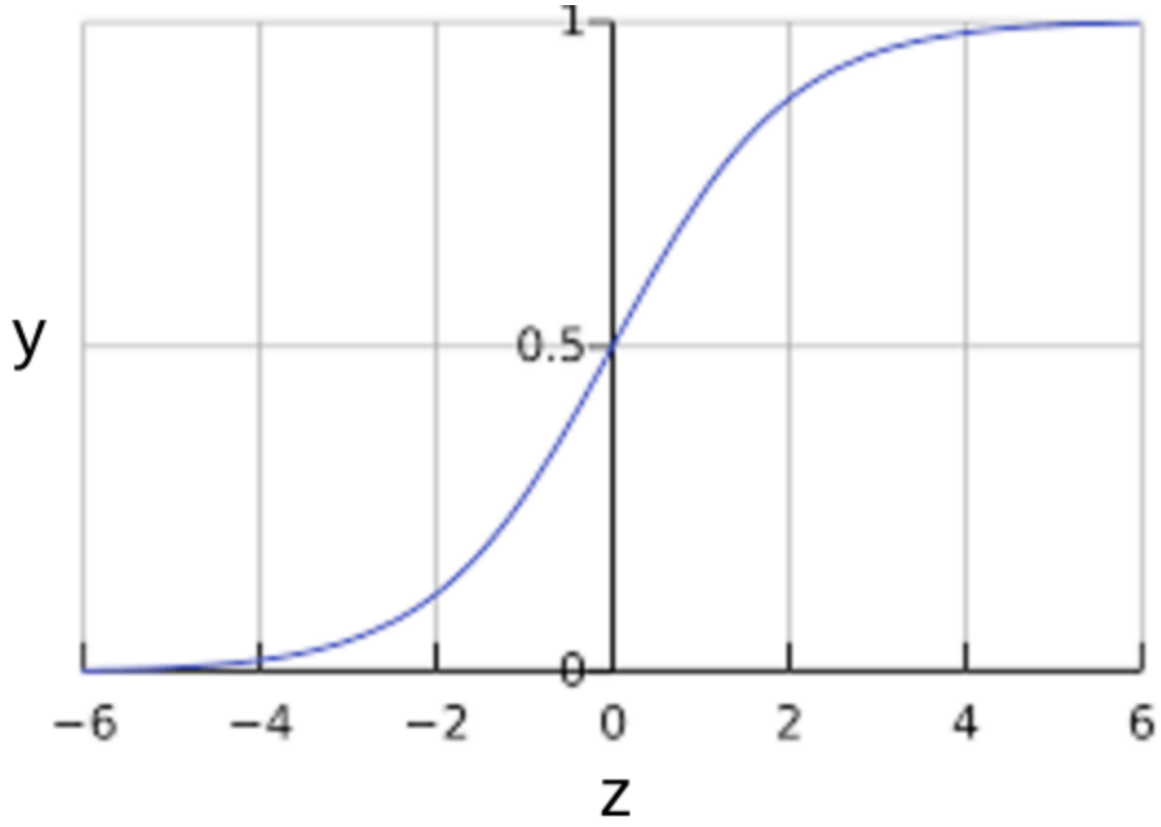

Figure 4.1: logistic_regression (sigmoid_activation_function)

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 4.2: overview_of_polynomial_regression_model

4.7 Method 4: 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.

4.8 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

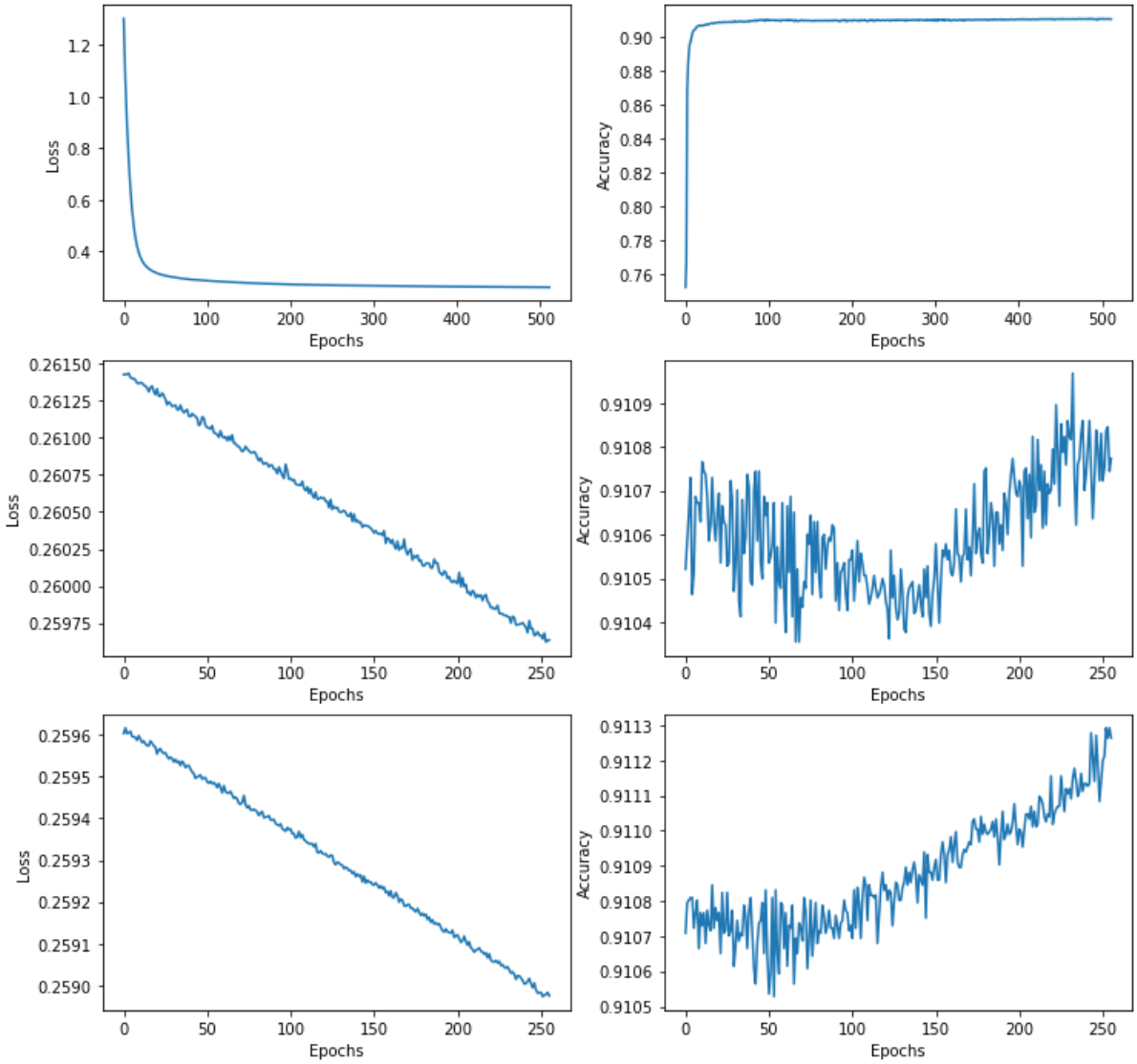

Figure 1: test_image_caption

Discussion of Results

Discussion goes here

Conclusion

Conclusion goes here

References