MTECH KE-5206 (CI1) PROJECT REPORT

1 CA Assignment 1 - Neural Network Ensembles

TEAM MEMBERS

SIEW YAW HOONG – E0267855
CHEW ZE YONG – E0267857
CHEUNG KA KEI – E0267847
ONG JUN XIANG – E0267370
QUEK WEI JIE – E0269774

MASTER OF TECHNOLOGY IN
KNOWLEDGE ENGINEERING
BATCH KE-30(2018)

Executive Summary

This paper presents two halves of a Neural Network problem: Classification problem, which is allotting argument inputs to a class amongst a predefined repertoire, and Regression problem, the prediction of a continuous target variable. Our problems and datasets have shown there are target values to work towards to in both cases, which are effectively supervised problems.

Classification problem

To determine from a dataset of daily usages of heavy Scania trucks, 170 anonymised input variables were collected, with the objectives of predicting the binary outcome of the Air Pressure System (APS) failure. "Positive" class failures are classified as related to the APS, while "Negative" denotes otherwise.

This problem was tackled via classification modelling with neural networks and ensemble methods. 5000 entries were randomly sampled from the training data, and 1000 entries form the testing data.

The objective of the classification is to predict the failure types and minimise the total misclassification, as a false positives leads to unnecessary maintenance (10 cost units), and false negatives results in an engineering defect (500 cost units). For this data set, Multilayer Perceptron with Backward Propagation performed the best in terms of increasing sensitivity and reducing penalty cost. Ensemble models has shown no significant improvement in accuracy and sensitivity over individual models.

Caret package in R is a great wrapper for working with many types of classification and regression models. However, it has its limitations as it does not support all functions fully, e.g.: has no probability function for Extreme Learning Machine. We can customise our own function for Caret, but it rendered the model inoperable in CaretEnsemble as it does not work with custom models. Radial Basis Function crashed when called from within Caret. Keras packages returned very low accuracies.

Regression problem

The regression problem explores predicting the number of comments a Facebook post can generate with the features and history of the particular post, using a combination of Pearson Correlation and model validation as data cleaning process. Thereafter, the neural network models trained on the cleaned data offers better prediction results..

With the GRNN being sensitive to large variances between the input features, transformation of the data by thenormalization of all features is required before training, On the contrary, the MLFF model does not require further transformations or normalization.

The GRNN model features a one-pass learning approach with only 1 hyperparameter that can be tuned. Although its training time is relatively efficient, the prediction is slightly slower when unseen data is input. On the other hand, the MLFF network, with many tuneable hyperparameters,takes much longer to optimise but iscompensated with a quick prediction time with unseen data. It is to note that both models performed satisfactorily.

As the ultimate goal is to provide an accurate prediction, several ensemble methods were used as a means to reduce the prediction errors over the same set of inputs.

In conclusion, model optimisation is an iterative process, many other aspects can be explored to realize a more powerful neural network model. These include the addition of relevant features with high predictive power, selection of better models, combination of different models and ensemble as well as customization based on domain knowledge.

Presentation Format

For clarity sake, this paper will present the APS classification problem first, followed by the Facebook comments regression problem.

Table of Contents

List of Figures

List of Tables

CLASSIFICATION PROBLEM: SCANIA TRUCKS APS

2 Background

For classification with neural network, the Scania Truck APS System data set was chosen (https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks).

The data was collected from heavy Scania trucks from daily usage. 170 anonymised input variables are collected, with the objectives of predicting the binary outcome of the Air Pressure System (APS) failure. "Positive" class attributes failure to the APS, and "Negative" class denotes otherwise. The objective of the classification is to predict the failure types and minimise the total misclassification.

In addition, Type I error (false positive) has a cost of 10 units, and Type II error (false negative) 500 units. Failure to predict an APS fault will lead to unscheduled breakdown and heavy repair cost, thus tagged to a higher penalty. The evaluation of the models is based on the minimising the penalty score which is determined by (10 x No. of False Positive) + (500 x No. of False Negative).

To reduce the number of false negatives, the sensitivity of the prediction can be maximised.One way of doing that is by modelling the problem as a regression problem, and manually adjust the trigger rate for the output. However, for the scope of this exercise, we will limit our techniques to classification modelling with neural networks and ensemble methods.

The 170 input variables consist of integer and numerical values, and the target output is categorised with 2 levels. In the original data set, there are 60,000 entries in the training data and 16,000 entries in the testing data. To improve computational time, 5000 entries were randomly sampled from the training data, and 1000 entries from the testing data.

2.1 Tools

- Caret in R Programme were the main tools used, supported by various packages.
- For creating individual neural net models:
    - Feed Forward Neural Network with Single Hidden Layer (nnet in nnet)
    - Multi-layer Perceptron with Back Propagation (mlpML in RSNNS)
    - Extreme Learning Machine (elm in elmNN)
- As the elm probability function has to be customised to run in Caret, we are not able to use CaretEnsemble to create the ensemble model as it doesn't work with custom model. So manual ensemble was done instead.

3 Procedure

3.1 Pre-Processing

As embarking with any data problems, pre-processingof the data forms a critical first step.. In this project, the training and testing data sets were imported as dataframes and the data which were not applicable were later defined. The dataframes were subsequently inspected for:

- Proportion of ‘pos’ and ‘neg’ in output data
- Missing data with ‘NA’
- Structure of input variables

For rapid solutioning, the reduced-size, stratified random sample data set was used. The following steps listed below were the initialization of data training process:

- Set the output column to factor.
- Remove columns with constant numbers.
- Impute NA by using k Nearest Neighbour (kNN).
- Normalise the data range of all columns between 0 and 1.
- Inspect the density plot matrix between all input and output variables to understand their relationship.

Optional – Feature selection using recursive feature elimination with random forest and k-fold cross validation.

3.2 Individual Neural Network Model

Three types of individual networks were tested with their accuracies compared: Feed Forward Neural Network with Single Hidden Layer, Multi-layer Perceptron with Back Propagation, and Extreme Learning Machine.

3.2.1 Feed Forward Neural Network with Single Hidden Layer

- From Caret, called ‘nnet’, and used ‘nnet’ as method and ROC as metric.
- Train the model with the training data
- Check for model accuracy and variable importance
- Predict the output with testing data
- Create the confusion matrix
- Plot the neural net diagram

3.2.2 Multi-layer Perceptron with Back Propagation

- From Caret, called ‘RSNNS’, and used ‘mlpML’ as method and ROC as metric
- Repeat the rest of the procedure in 2.2.1

3.2.3 Extreme Learning Machine

- From Caret, called ‘elmNN’, and used a manually adjusted ‘elm_fun’ as method and ROC as metric.
- Repeat the rest of the procedure in 2.2.1

4 Compare the accuracy of the individual models

4.1 Ensemble Neural Network Model

The ensemble model was formed with other learning algorithms and using their predicted results, a mixer algorithm was put through the training process to make a final prediction using the predictions of the other algorithms as additional inputs.

The main technique used in this section is Stacking, 3.3.1 and 3.3.2 forms the two-step process which prepares the data for learning.

4.1.1 Averaging

Extract the probability values of the predicted output from the 3 individual models. Compute the average for each row and convert it to factor of 0 or 1 based on a 0.5 cut-off.

4.1.2 Majority Voting

The binary outputs from the 3 individual models are tabulated, and the majority for each row becomes the output

4.1.3 Weighted Average

We can apply weights to different models base on their accuracy. However, this was not done as the models are observed to have similar accuracies.

4.1.4 Weighted Voting

We can apply weights to different models base on their accuracy. However, this was not done as the models are observed to have similar accuracies.

4.1.5 Ensemble Stacking Model

1. Out-of-fold prediction probabilities of each individual model are extracted from the training data. These will become the training data for the stacking model.
2. Out-of-fold prediction probabilities of each individual model are extracted from the testing data. These will become the testing data of the stacking model.
3. Stochastic Gradient Boosting (GBM), Generalised Linear Model (GLM), and Random Forest (RF) models are created, with the results from Step 2 as the training data, and the original training output as the output
4. The new GBM, GLM and RF models are used to predict the output, with the results from Step 2 as the testing data.

5. The results from Step 4 are compared with the original testing data set output to obtain the confusion matrix for the ensemble models.

4.1.6 Compare the accuracy of the ensemble models.

5 Results

5.1 Individual Neural Networks

5.1.1 Feed Forward Neural Network with Single Hidden Layer

- 5-fold cross-validation was performed at training
- ROC was used to select the optimal model from 5 options of hidden nodes and 5 options of weight decay
- Optimal model has hidden node number of 5 and weight decay of 0.1

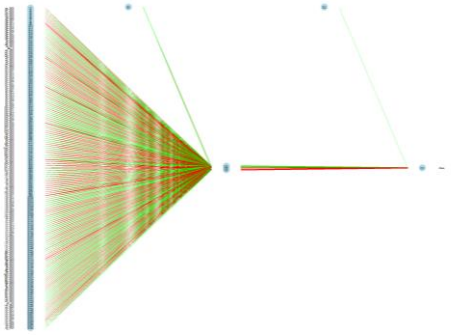


Figure 1 : NNet Diagram

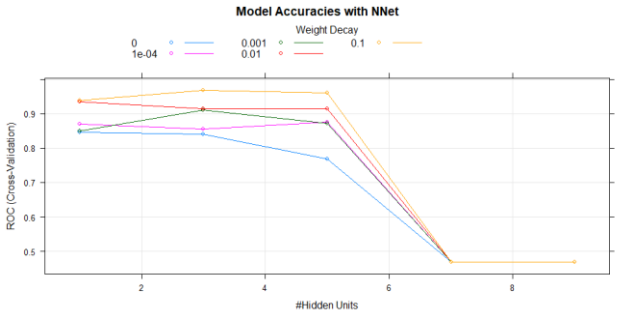
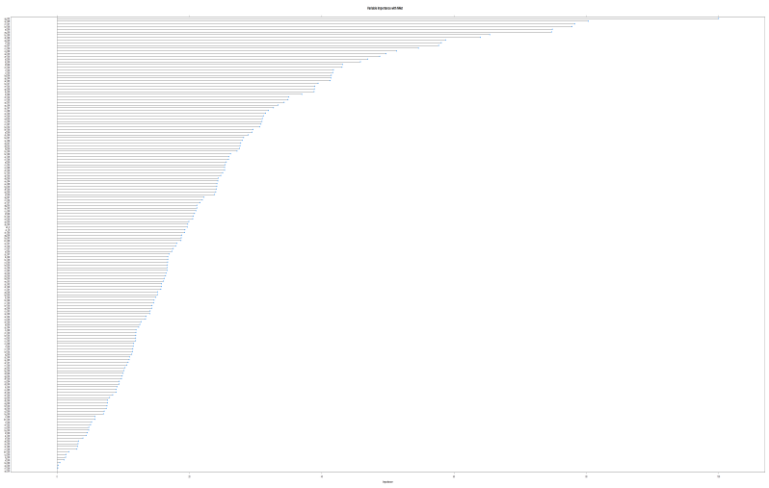


Figure 2 : Model accuracies with different hidden nodes



Top 5 Variables:

- ag_002
- ay_009
- cn_001
- ba_008
- ar_000

Figure 3 : Variable importance

Actual / Predicted	Neg	Pos
Neg	975	10
Pos	2	13
Accuracy = 0.988 **Sensitivity = 0.565**		

Table 1 : Confusion Matrix

5.1.2 Multi-layer Perceptron

- 5-fold cross validation was performed at training
- ROC was used to select the optimal model from 3 options of hidden nodes for 3 hidden layers
- Optimum model for 3 hidden layers:
    - Layer 1 – 60 nodes
    - Layer 2 – 30 nodes
    - Layer 3 – 10 nodes

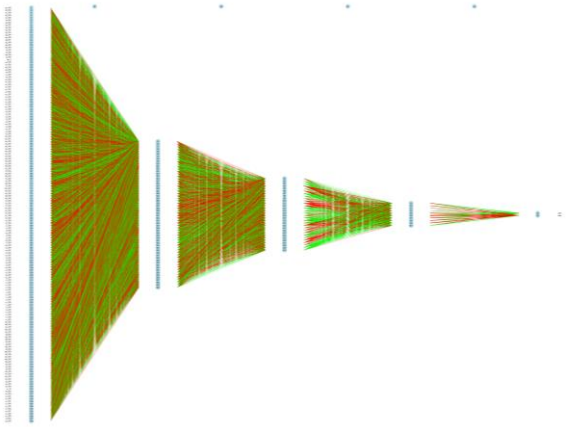


Figure 4 : MLP Diagram

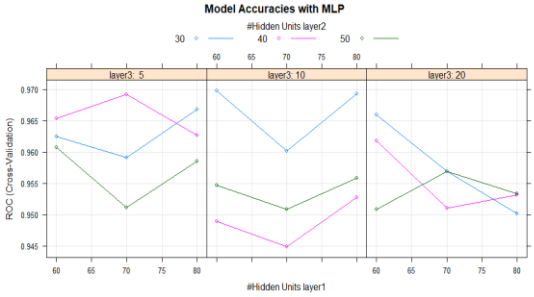
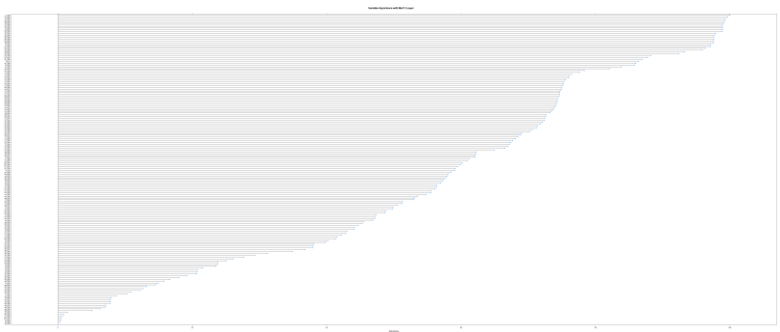


Figure 5 : Model accuracies with different hidden layers and nodes



Top 5 Variables:

- ci_000
- ck_000
- bj_000
- ap_000
- aa_000

Figure 6 : Variance Importance

Actual / **Predicted**	**Neg**	**Pos**
Neg	971	6
Pos	6	17
Accuracy = 0.988 **Sensitivity = 0.739**		

Table 2 : Confusion Matrix

5.1.3 Extreme Learning Machine

- 5-fold cross validation was performed at training
- ROC was used to select the optimal model from 4 options of hidden nodes and 4 options of activation functions
- Optimal model has 20 hidden nodes of "sine" activation function
- Extreme Learning Machine has the least processing time during model processing. However, it also has the lowest total score. Also noteworthy was that the probability spread of the output prediction is restricted to a smaller range (0.2 to 0.7), as opposed to the other models where the probability spread across 0 to 1.

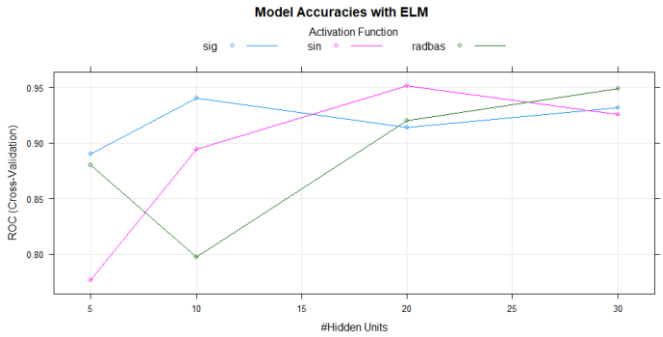
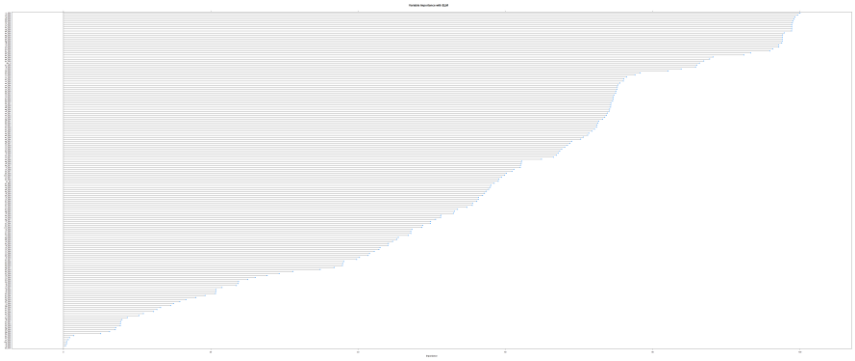


Figure 7: Model accuracies with different hidden nodes



Top 5 Variables:

- ci_000
- ck_000
- bj_000
- ap_000
- aa_000

Figure 8 : Variable Importance

Actual / Predicted	Neg	Pos
Neg	973	20
Pos	4	3
Accuracy = 0.976 **Sensitivity = 0.130**		

Table 3 : Confusion Matrix

6 Model Comparison

- ROC, Sensitivity and Specificity comparison

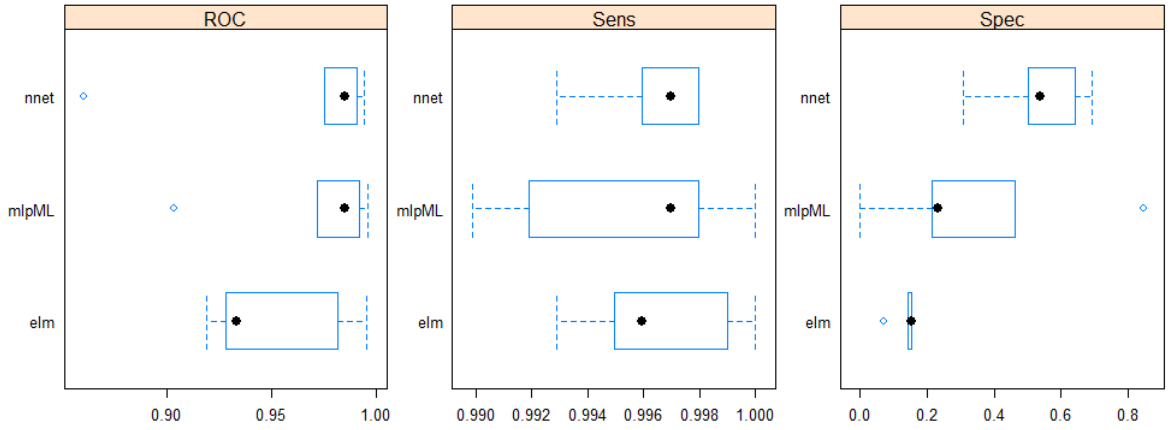


Figure 9 : ROC, Sensitivity and Specificity comparison

- Correlation comparison
  - As the models have low correlation, they are suitable for ensemble techniques to improve the predictions

```
•                nnet        mlpML        elm
•  nnet   1.000000000 -0.006275453  0.5550783
•  mlpML -0.006275453  1.000000000 -0.5691549
•  elm    0.555078268 -0.569154906  1.0000000
```

6.1 Ensemble Neural Networks

Averaging

Actual / Predicted	Neg	Pos
Neg	976	12
Pos	1	11
Accuracy = 0.987		
Sensitivity = 0.478		

Majority Voting

Actual / Predicted	Neg	Pos
Neg	976	12
Pos	1	11
Accuracy = 0.987		
Sensitivity = 0.478		

6.2 Stacking with Stochastic Gradient Boosting Stacking with Logistic Regression

Actual / Predicted	Neg	Pos
Neg	976	14
Pos	1	9
Accuracy = 0.985 **Sensitivity = 0.391**		

Actual / Predicted	Neg	Pos
Neg	976	14
Pos	1	9
Accuracy = 0.985 **Sensitivity = 0.391**		

6.3 Stacking with Random Forest

Actual / Predicted	Neg	Pos
Neg	973	14
Pos	4	9
Accuracy = 0.982 **Sensitivity = 0.391**		

7 Summary

	Individual Models			Ensemble Models				
	NNET	**MLP**	**ELM**	**Average**	**Voting**	**Stacking GBM**	**Stacking GLM**	**Stacking RF**
Accuracy	0.988	0.988	0.976	0.987	0.987	0.985	0.985	0.982
Sensitivity	0.565	0.739	0.130	0.478	0.478	0.391	0.391	0.391
Penalty Cost	5020	3060	10040	6010	6010	7010	7010	7040

Penalty Cost = (10 x No. of False Positive) + (500 x No. of False Negative)

8 Conclusions and Learning

1. For this data set, Multilayer Perceptron with Back-Propagation performed the best in terms of increasing sensitivity and reducing penalty cost.
2. Ensemble models has shown no significant improvement in accuracy and sensitivity over individual models.
3. Caret package in R is a great wrapper for working with many types of classification and regression models. However, it has its limitations as it does not support all functions fully, eg:
    - It has no probability function for Extreme Learning Machine. Customisation of functionsfor Caret is possible, however it will render the model inoperable in CaretEnsemble as it does not work with custom models.
    - Radial Basis Function crashed when called from within Caret
    - Keras packages returned very low predictions.

REGRESSION PROBLEM:

PREDICTING FACEBOOK COMMENTS COUNT

1 Background

1.1 Preface

As social networking service becomes increasingly used, the amount of data uploaded to them has dramatically risen. This generated information provides a way to study and understand the behaviour of its users. With the ability to predict the number of comments a Facebook post can generate with the features and history of the particular post, a targeted approach can be devised to sieve out content which may be of interest to large groups of people.

1.2 Dataset

The "Facebook Comment Volume Dataset" is a multi-variate dataset consisting of 40,949 instances of 53 features with 1 target variable. The task is to perform a regression on the features to predict the number of comments that a particular post will receive in a pre-defined duration.

The dataset composition is shown in Table 1.

Table 4: Composition of Dataset

Feature No.	**Feature Name**	**Feature Type**	**Description**
1	Page Popularity/likes	Page feature	Defines the popularity or support for the source of the document.
2	Page Checkin	Page feature	Describes how many individuals so far visited this place. This feature is only associated with the places eg:some institution, place, theater etc.
3	Page talking about	Page feature	Defines the daily interest of individuals towards source of the document/ Post. The people who actually come back to the page, after liking the page. This include activities such as comments, likes to a post, shares, etc by visitors to the page.
4	Page Category	Page feature	Defines the category of the source of the document eg: place, institution, brand etc.
5 – 29	Derived	Derived feature	These features are aggregated by page, by calculating min, max, average, median and standard deviation of essential features.

30	CC1	Essential feature	The total number of comments before selected base date/time.
31	CC2	Essential feature	The number of comments in last 24 hours, relative to base date/time.
32	CC3	Essential feature	The number of comments in last 48 to last 24 hours relative to base date/time.
33	CC4	Essential feature	The number of comments in the first 24 hours after the publication of post but before base date/time.
34	CC5	Essential feature	The difference between CC2 and CC3.
35	BaseTime	Other feature	Selected time in order to simulate the scenario.
36	Post length	Other feature	Character count in the post.
37	Post Share Count	Other feature	This features counts the no of shares of the post, that how many peoples had shared this post on to their timeline.
38	Post Promotion Status	Other feature	To reach more people with posts in News Feed, individual promote their post and this features tells that whether the post is promoted(1) or not(0).
39	H Local	Other feature	This describes the H hrs, for which we have the target variable/ comments received.
40-46	Post published weekday	Weekdays feature	This represents the day(Sunday...Saturday) on which the post was published.
47-53	Base DateTime weekday	Weekdays feature	This represents the day(Sunday...Saturday) on selected base Date/Time.
54	Target	Target variable	The no of comments in next H hrs (H is given in Feature no 39).

1.3 Tools Used

Python 3.6 was used as the tool of choice to perform the regression analysis on the dataset. In particular, two types of Neural Network (NN) architectures were used to conduct the analysis, namely, Multi-Layer Feed Forward (MLFF) NN and General Regression Neural Network (GRNN).

The Python packages used for the MLFF and GRNN were Keras and Neupy respectively.

2 Data Cleaning

Firstly, the headers of each column were appended to the dataset as they were not included in the CSV file.

Subsequently, the dataset was explored for inconsistencies and irrelevancies. It was found that there were many instances where the feature “CC1” is zero. A zero “CC1” can be interpreted as a post with its comment function disabled. These instances were removed as the target variable will always remain as zero without the comment function enabled.

Next, it was discovered that instances with “BaseTime” more than 48 hours were almost invariant, as compared with those with “BaseTime” less than 48 hours. This can be seen from the respective scatter plots depicting the distribution of the comment volumes below:

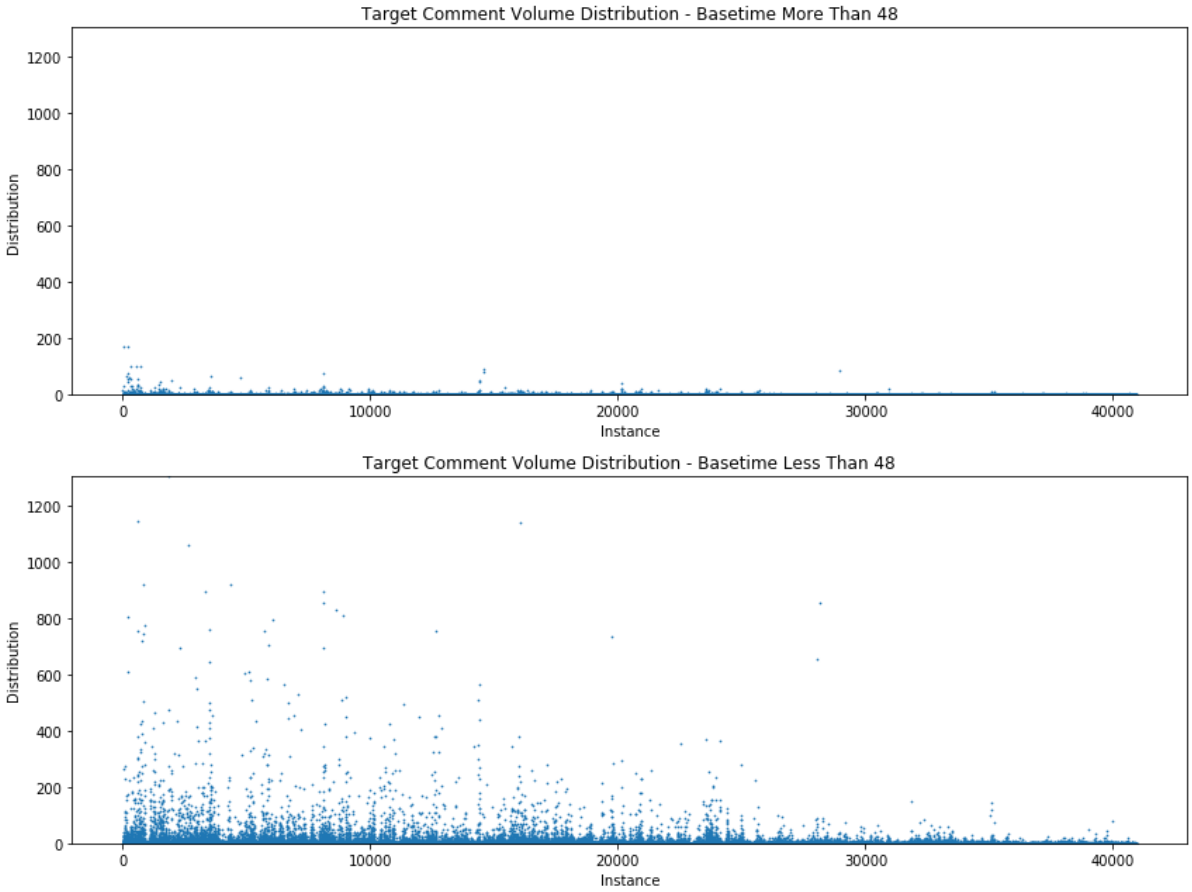


Figure 10 : Comparison between Comment Distribution with BaseTime > 48 hours (top) & BaseTime ≤ 48 hours (bottom)

3 Feature Selection

After removing the irrelevant features, Pearson Correlation (PC) was performed on the remaining instances of the dataset.

The results of the PC show the degree of correlation with the target variable, as well as the inter-correlation between the features. A generated Heatmap portrays the correlation intensity of each variable, coloured according to the values of PC.

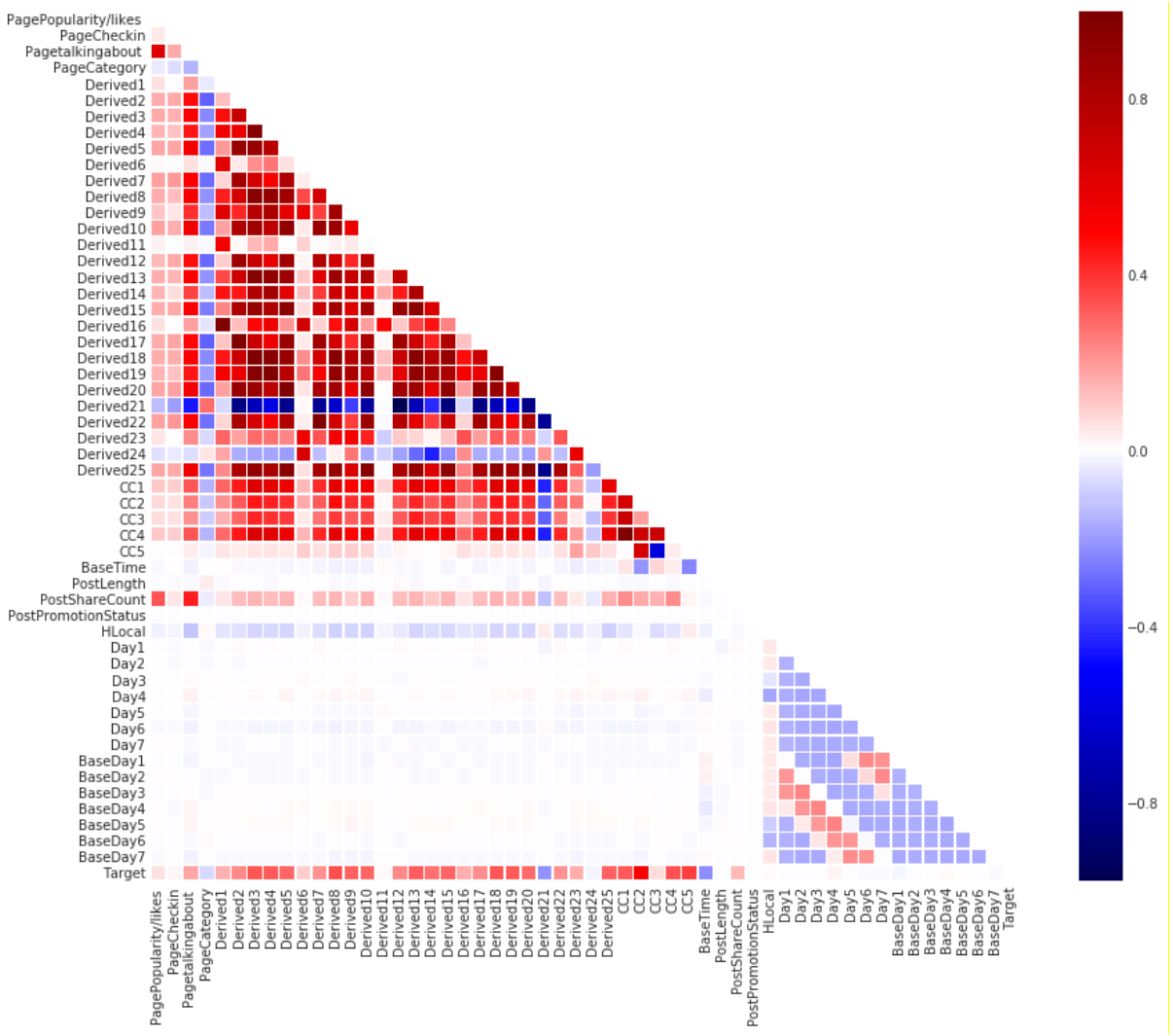


Figure 11 : Pearson Correlation Heatmap

The Heatmap has provided several useful insights.

Firstly, the features with a correlation score of lesser than 0.2 and more than -0.2 with respect to the “Target” column (represented by light colour grid) were removed as these features were deemed to have low predictive power.

Next, many of the “Derived #” and “CC #” features share high PC values with each other, as illustrated by its corresponding colour intensities. Among these highly correlated features, only those with the highest PC score were retained. The remaining features were removed from the dataset as they do not carry additional information and would not serve as useful predicates.

With reference to the “Target” row of heatmap, the features “Day #” and “BaseDay #” are categorical features which were encoded into 7 binary featuresusing One-Hot Encoding (OHE). As such, its PC scores were not good indicators of their usefulness to the neural network

prediction. To validate their usefulness, a comparison was made based on their effect on the model performance.

A single layer MLFF network was trained and validated using these features, and the figure below shows that the addition of "Day #" and "BaseDay #" has minimal impact on the model performance (Orange lines of the graphs in Figure *12*). As such, these features were dropped from the dataset.

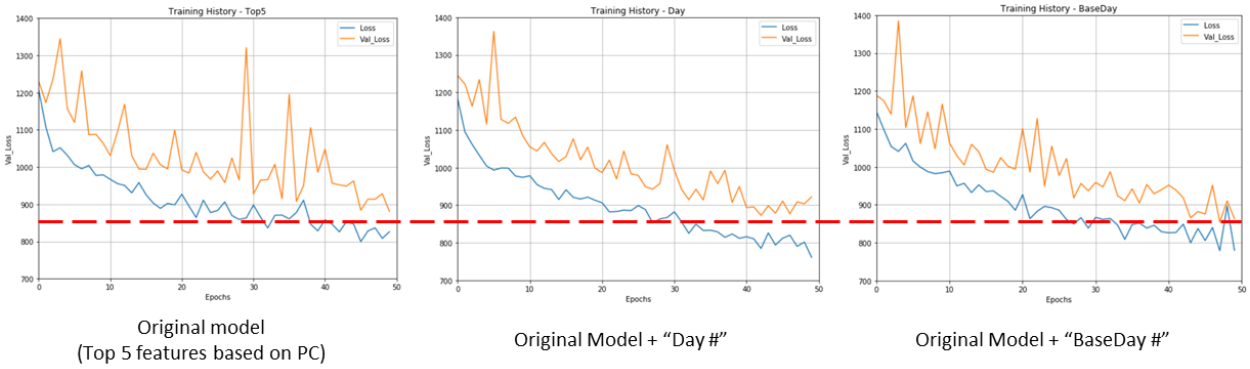


Figure 12 *: Model Performance Comparison based on Training History*

The remaining columns after the feature selection exercise were "CC2", "Derived8", "PostShareCount", "BaseTime" and the "Target" variable.

4 General Regression Neural Network (GRNN)

GRNN was used as the first method of performing the regression. The GRNN architecture consist of a series of input neurons, which are fully connected to a hidden layer which is called the "Pattern Layer". The pattern layer utilizes a Gaussian function as the activation for firing the neurons.

The outputs of the hidden nodes are subsequently pumped into the summation neurons which performs a regression on the target variable based on the input nodes.

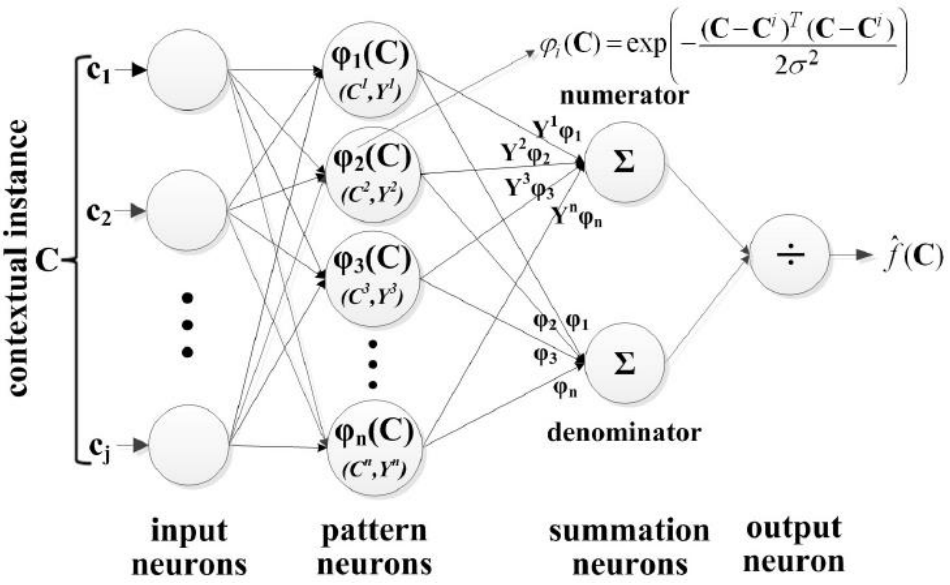


Figure 13: GRNN Architecture

4.1 GRNN Results

The GRNN model was initially put through an iterator to find its best Sigma value. The model achieved the lowest Mean Squared Error, MSE at $\sigma = 0.13673$.

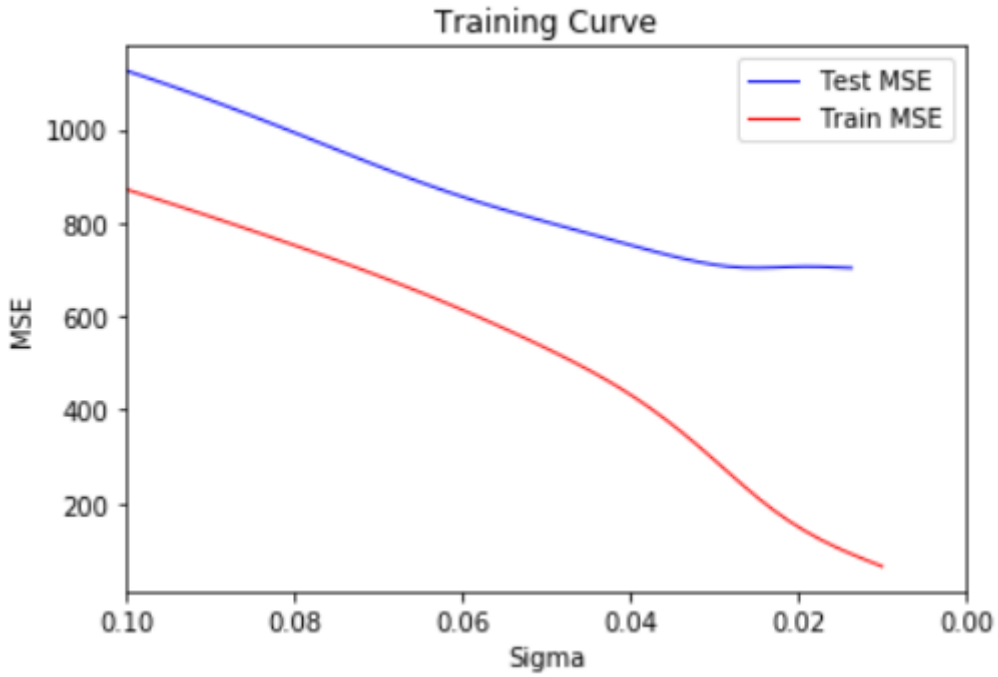


Figure 14: GRNN, MSE vs Sigma

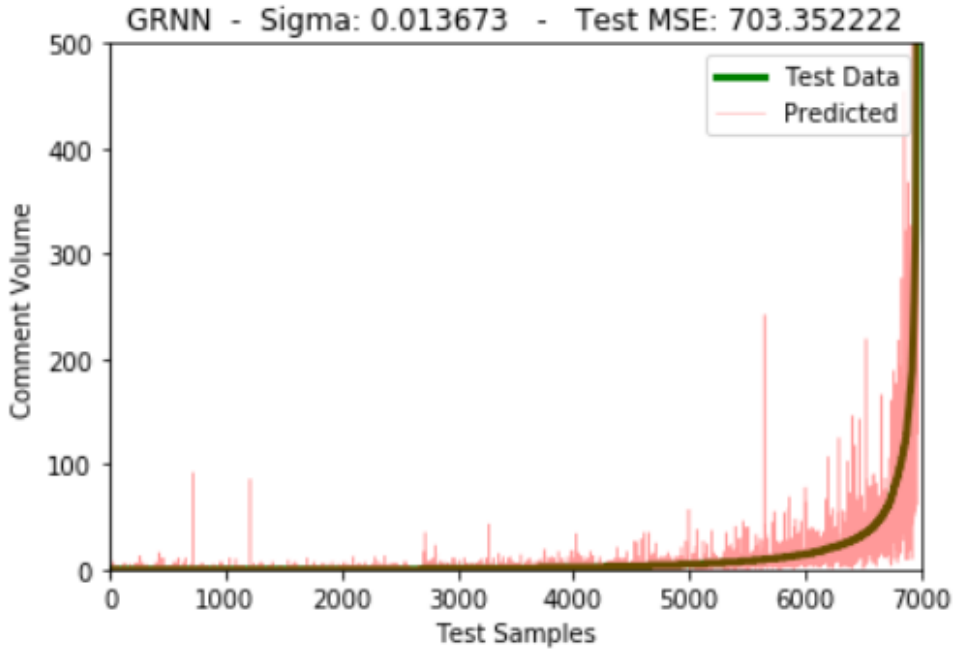


Figure 15: GRNN, Lift Curve

The GRNN performed exceedingly well for data that produces a low comment volume. However, as the comment volume starts to increase, the prediction accuracy starts to deviate to a large extent.

This phenomena can be due to the inherent data structure, as instances of seemingly similar inputs can produce a large discrepancy in the corresponding outputs.

5 Multi-Layer Feed Forward Neural Network (MLFF)

MLFF was used as the second method of performing the regression. The MLFF architecture used consist of a series of input neurons, which are fully connected to a set of hidden layers with varying numbers of neurons. One of the popular learning techniques of MLFF is back-propagation, where the difference between MLFF's output and actual value of target variable (error) is fed backwards through every hidden layers to adjust the weight of neurons.

An illustration of MLFF network is shown below:

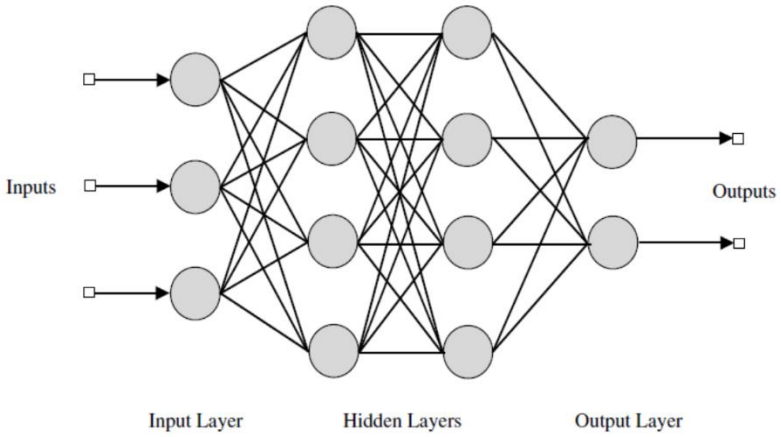


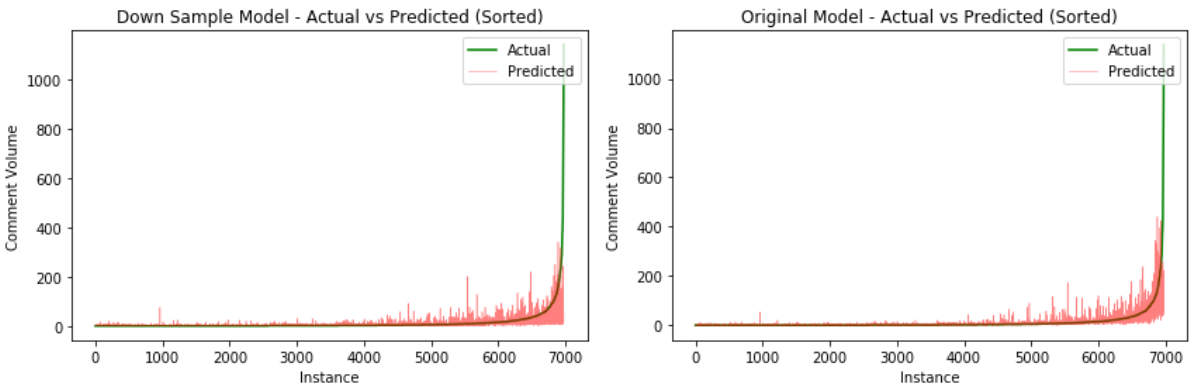
Figure 16 : Architecture of MLFF Networks

5.1 MLFF Results

To drive down the MSE of a MLFF network, several hyperparameters need to be tuned. The tuning of MLFF hyperparameters was conducted using the GridSearchCV (exhaustive) or RandomizedSearchCV (randomized, with computational budget specified) of Scikit-Learn.

As these search processes are computationally intensive, techniques to reduce the computational requirement were needed.

The first approach was to reduce the size of the dataset. The original dataset was down-sampled by 5 times to about 8,000 instances. In order to ensure the down-sampled data is representative of the original dataset, a comparison of model trained using the down-sampled data and original data was done. The performance of these models was comparable, as shown in their respective lift curves below:




```
Original Model MSE =  1082.1916140265228
Down Sample Model MSE =  1347.7867925237583
```

Figure 17 : Comparison between Model trained on Down-sampled dataset (Left) Model trained on Original dataset (Right)

Next, a GridSearch was carried out with the individual hyperparameters, while keeping the other hyperparameters constant to determine the top options of hyperparameters to be used. The hyperparameters tuned were optimizer, activation function of the hidden layers, the learning batch size, the number of hidden layers and neurons, and the network initialization method.

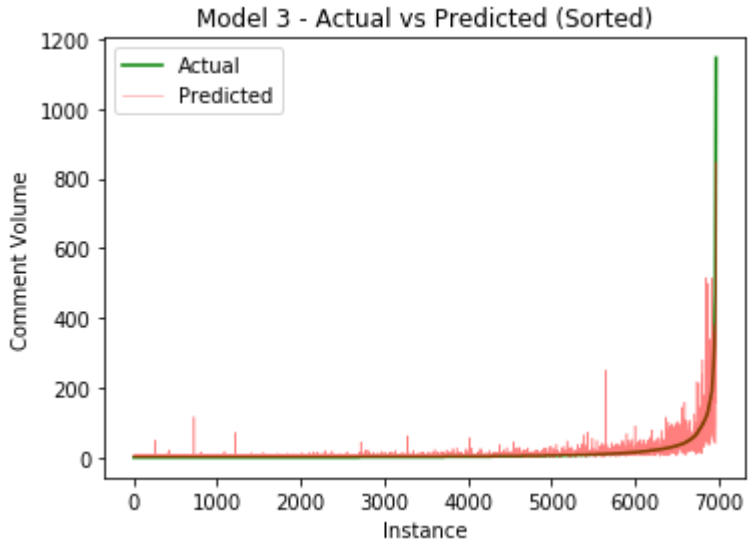
A sample output for the tuning of optimizer is shown below, where the top optimizers 'Adadelta', 'Adagrad' and 'Adamax' were identified for subsequent model tuning.

```
Best: -592.608084 using {'optimizer': 'Adadelta'}
-889.442504 (215.601692) with: {'optimizer': 'SGD'}
-620.454128 (231.090728) with: {'optimizer': 'RMSprop'}
-608.358948 (214.708310) with: {'optimizer': 'Adagrad'}
-592.608084 (219.712238) with: {'optimizer': 'Adadelta'}
-626.369506 (232.932432) with: {'optimizer': 'Adam'}
-606.059558 (215.556846) with: {'optimizer': 'Adamax'}
-766.069677 (430.028212) with: {'optimizer': 'Nadam'}
```

Figure 18 : Tuning of MLFF Optimizers

Once a range of better hyperparameters were identified, a RandomizedSearch was carried out with all these hyperparameters, to identify the best combination of hyperparameters to train the MLFF model on.

The lift curve of the optimized MLFF model is similar to that of the GRNN model, with a MSE of 689.10.




```
Model 3 MSE =  689.1049526011842
```

Figure 19 : Optimized MLFF Model (Model 3)

6 Neural Network Ensemble

In order to further reduce the MSE, several ensemble approaches were tested: the GRNN ensemble, the MLFF ensemble and the GRNN + MLFF ensemble.

6.1 GRNN Ensemble

An ensemble approach to aggregate the results of 4 GRNN networks was utilized.

The training dataset was split into 4 partitions, each comprising of about 4067 instances. The features selected was the same as with the non-ensemble method.

The architecture of the GRNN ensemble is shown below.

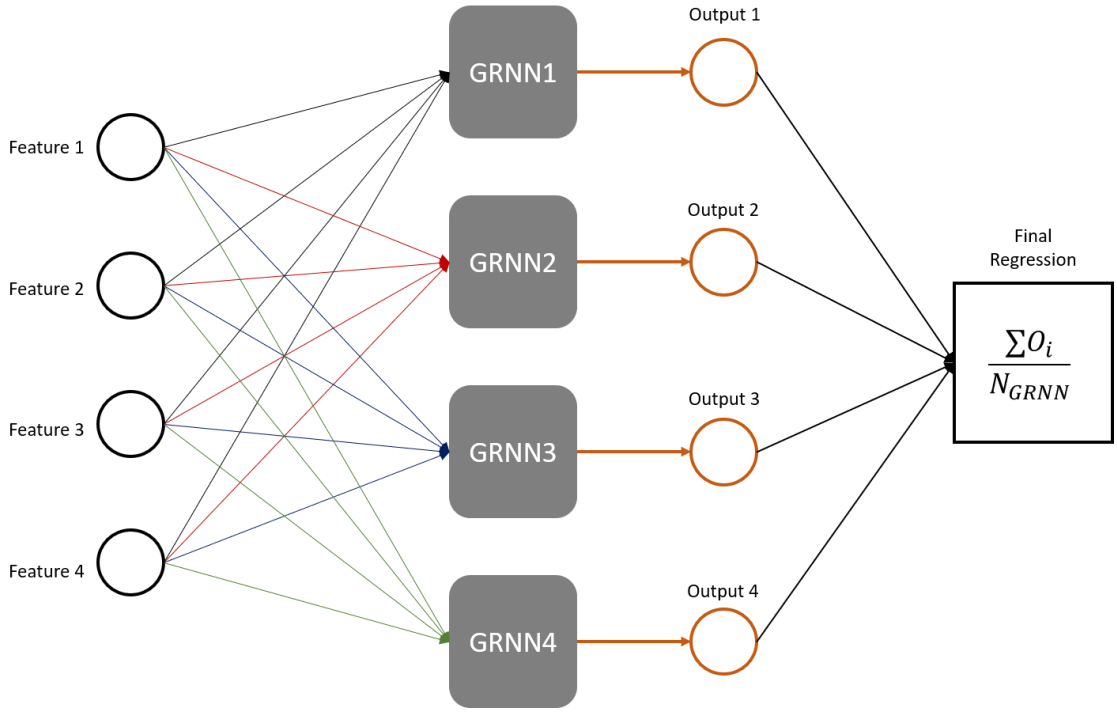


Figure 20: GRNN Ensemble Architecture

6.2 GRNN Ensemble Results

Each GRNN model was then put through the iterator to find its best Sigma value. The best corresponding Sigma value was selected for use in each model of the ensemble.

Table 5: First 5 rows of Sigma iteration for each GRNN

Partition	Sigma	MSE Train	MSE Test	MAE Train	MAE Test
1	0.1	704.0442819	1193.311276	8.135714398	9.624216639
1	0.098163265	690.2803708	1182.606883	8.068476223	9.572727175
1	0.096326531	676.2684284	1171.904966	7.999783902	9.520833549
1	0.094489796	662.0580438	1161.221658	7.929468749	9.467703163
1	0.092653061	647.7067961	1150.571922	7.859622585	9.415913328

Table 6: First 5 rows of GRNN Ensemble predictions & squared-errors

GRNN1	GRNN2	GRNN3	GRNN4	Averaged_GRNN	Target	Sq_Err
1.2484	1.3374	1.4444	1.5685	**1.3997**	0	1.9590
1.0152	1.2706	1.3052	1.4481	**1.2598**	4	7.5088
1.1221	1.1911	1.3882	1.4677	**1.2923**	0	1.6700
1.6704	1.7017	1.7449	1.4478	**1.6412**	0	2.6935
2.1894	2.0391	2.3982	2.2800	**2.2267**	1	1.5047

Each of the models in the GRNN ensemble produced a higher MSE compared to the baseline model even when using the best achievable Sigma value obtained from the iterator.

The squared-error of each prediction was derived using the average of the regressions performed by the 4 models. Subsequently, the squared-error was calculated from the difference between the prediction and the target output.

Finally, the MSE was determined from the average of the sum of squared-error data on all of the test instances. The MSE obtained via the ensemble approach is **862.05.**

The increased MSE for each of the GRNN in this scenario may be due to the distribution of the dataset. As there are significantly lesser data available for high comment volume, splitting the data into 4 parts effectively reduces the training data for them, leading to higher errors when regressing against the related inputs.

However, it is worthy to note that the ensemble approach outperformed the individual GRNN models train on the partitioned data.

6.3 MLFF Ensemble

The same set of hyperparameters were used for 5 runs of RandomizedSearch. The best combination of hyperparameters for each search is used to construct 5 separate MLFF models trained using the same set of data. The MSEs of these individual models range from **689.10** to 867.69.

```
Model 1 MSE =  867.6931687748369
Model 2 MSE =  814.1980784461782
Model 3 MSE =  689.1049526011842
Model 4 MSE =  858.4996985379454
Model 5 MSE =  820.6629811806957
```

Figure 21 : MSE of Individual MLFF Models

Next, similar to the GRNN ensemble architecture, the predictions of individual MLFF models were summed and averaged to obtain the aggregated prediction.

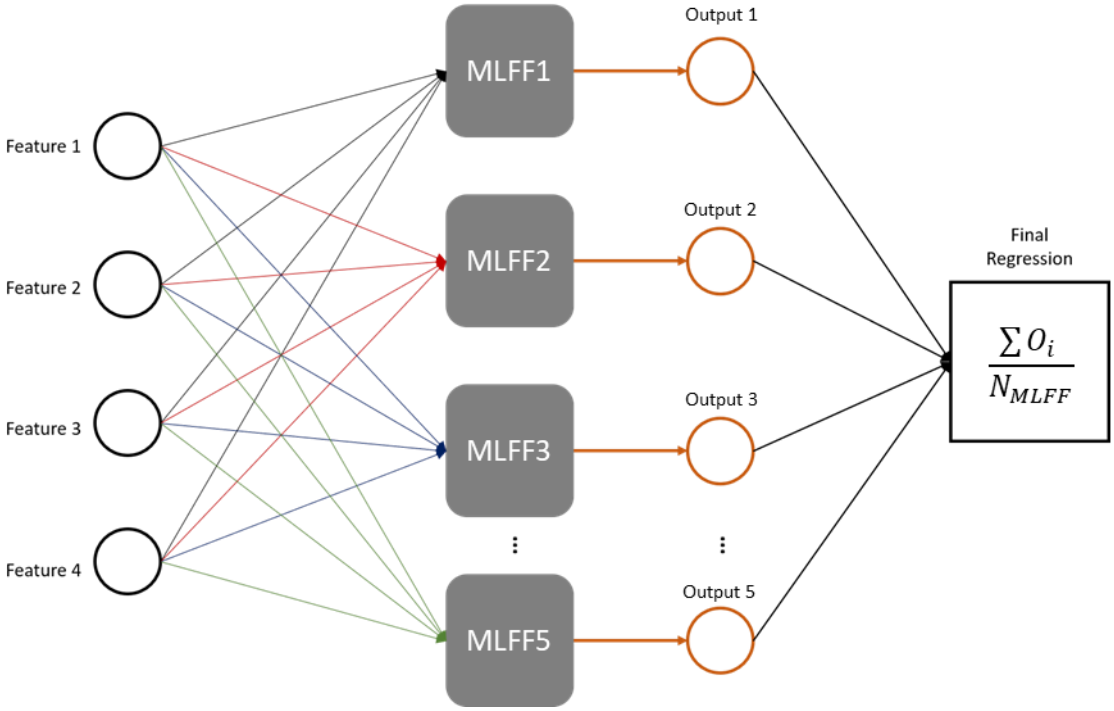


Figure 22 : MLFF Ensemble Architecture

In theory, averaging smoothens the inaccuracies in the outputs of individual models and results in better predictions across all test instances. However, due to the skewness in the distribution of the original dataset, majority of the trained MLFF models tend to have a larger prediction errors for test instances with which produces high comment volume.

As such, the ensemble's result for test instances with large actual values averages out at a lower predicted value, resulting in higher MSE of **779.16** as compared to the best performing individual MLFF's MSE of **689.10**, even though the predictions for test instances with low actual value performed better.

6.4 GRNN + MLFF Ensemble

To further enhance the prediction, a cross-models ensemble between GRNN and MLFF ensemble was developed. The predictions of GRNN and MLFF ensemble on the test data (30% of entire dataset) were treated as 2 additional features, and these were used as 2 extra inputs to train and test the GRNN + MLFF cross-ensemble. The architecture and training arrangement is shown below:

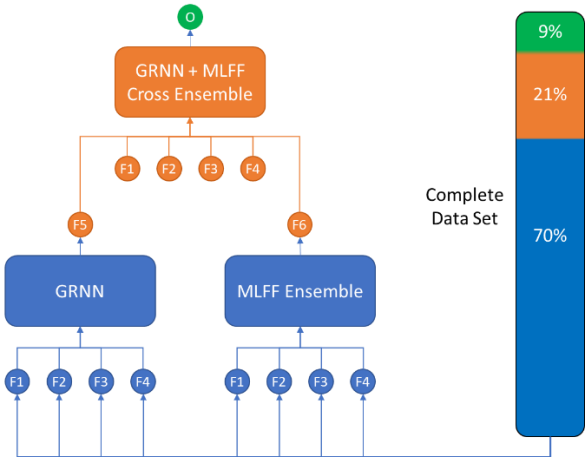


Figure 23 : Architecture and Training Arrangement of Cross-ensemble

The predictions given by the best GRNN model and MLFF ensemble served as useful gauges for the cross-ensemble. Consequently, a more accurate prediction should be provided by the cross-ensemble. This is analogous to reaching a better financial decision with the advice of top financial consultants who weigh in on their opinions.

For a fair comparison, the cross-ensemble should not be trained on data that has been 'seen' by the GRNN and MLFF ensemble. Therefore, only the data used to test the GRNN and MLFF ensemble can be used for the cross-ensemble training. As such, the cross-ensemble was only trained on 21% of the original data, as depicted in Figure 23.

To establish a fair performance assessment between the GRNN model, MLFF ensemble and the cross-ensemble, the 9 % of the data not 'seen' by all 3 types of model/ ensemble were used, with their respective MSE compared.

Despite being trained on a much lower amount of data (21% of the entire dataset), the cross-ensemble achieved the lowest MSE of **788.22**.

```
GRNN + MLFF Ensemble MSE =  788.2221355565331
GRNN Ensemble MSE =  919.7170234113712
MLFF Ensemble MSE =  818.828030925243
```

Figure 24 : Performances of Different Ensembles and Model

7 Conclusion

In the domain of machine learning, data cleaning is always the most crucial step that takes up close to 80% of data scientists' time (Press, 2016). By removing unnecessary features of the Facebook Comment Volume dataset using a combination of Pearson Correlation and model validation, the neural network models trained on the resultant dataset were better in prediction and took significantly lesser time to train.

Once the dataset cleaning was completed, it was necessary to transform the dataset according to the requirements of the network. For instance, GRNN is sensitive to high values of input features. Therefore, the data required normalization such that all the features are of similar scale, before they can be used to train the GRNN model. On the other hand, the cleaned data worked fine with MLFF model without further transformation or normalization.

In terms of the model's characteristics, GRNN featured a one-pass learning with only 1 hyperparameter to be tuned. Therefore, its training time is efficient, with the downside being a slightly slower prediction time when new data is passed to the network. On the other hand, the training of MLFF is a much longer process as there are many tuneable hyperparameters. This is compensated by a quick prediction time. Both models performed satisfactorily for this dataset.

As the ultimate goal is to provide an accurate prediction, a series of ensembles were used to smoothen the prediction errors over the same set of inputs.

In comparison with the previous classification data set, Multilayer Perceptron with Backward Propagation performed the best, in terms of increasing sensitivity and reducing penalty cost. Ensemble models, however, showed no significant improvement in accuracy and sensitivity over individual models.

Ultimately, model tuning and improvement is an iterative process, many other aspects can be explored to realize a more powerful neural network model. These include the addition of relevant features with high predictive power, selection of better models, combination of different models and ensemble as well as customization based on domain knowledge.