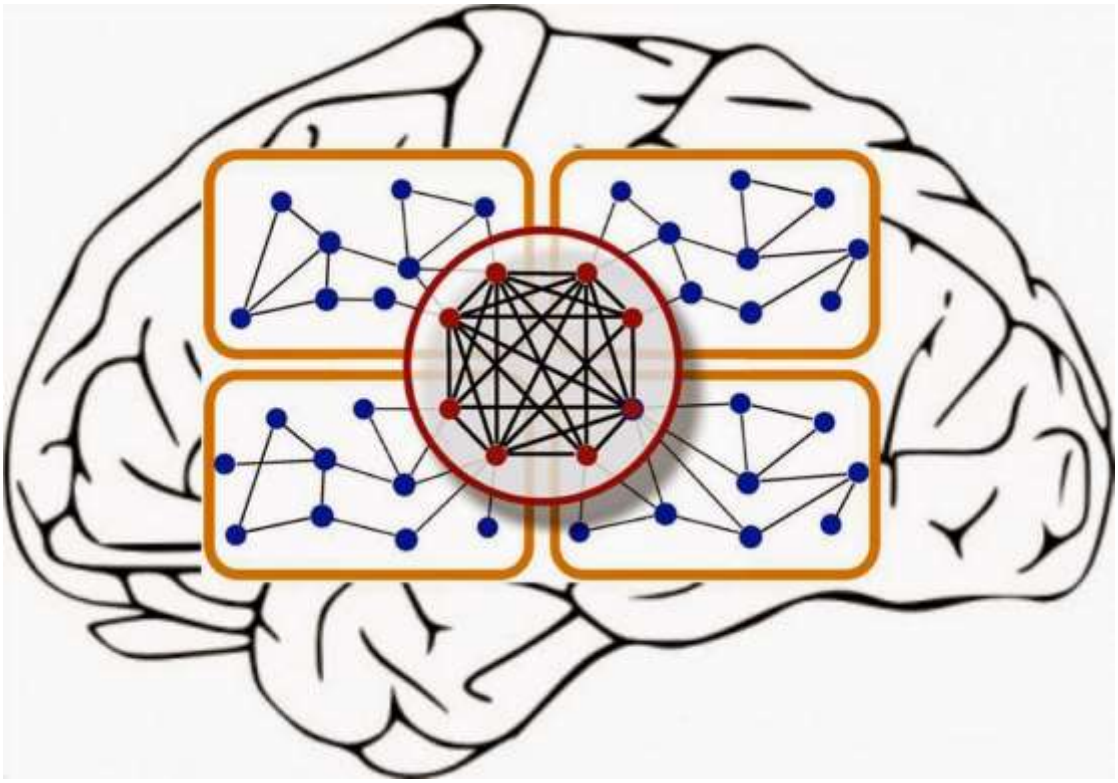
MTECH KE-5206 (CI-1) PROJECT REPORT

USING NEURAL NETWORKS FOR CLASSIFICATION AND REGRESSION



TEAM MEMBERS

ANURAG CHATTERJEE (A0178373U)
BHUJBAL VAIBHAV SHIVAJI (A0178321H)
CHAN YI JIE KELVIN (A0178430E)
KOH LAM SENG (A0179666H)
LIM PIER (A0178254X)

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

Contents

1. Executive Summary

Neural networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" (i.e. progressively improve performance on) tasks by considering examples, generally without task-specific programming.

We made use of various neural network architectures for a classification and a regression problem. For Classification we used the **Fashion MNIST** data set which is composed of 10 categories of article images from Zalando, a German fashion Ecommerce company. This data was obtained from Kaggle. The **Keras** library with a **TensorFlow** backend was used for developing the Neural network models. **Multi-layer feedforward Neural network with back propagation** (**MLFF-BP**), **Convolution Neural Networks (CNN)** and **Recurrent Neural Networks (RNN)** were implemented and their results on the same test data were compared against each other. We also tried **Capsule Networks** conceptualized by Geoffery Hinton [1]**.** In the end, these tested algorithms were **ensembled** together to give an ensemble model designed for this Fashion MNIST data set classification problem**.**

For Regression, we used an online News Popularity data set which was obtained from UCI [2]. The objective for the problem is to predict the number of “shares” of a specific Mashable article in social network sites. The **Scikit-Learn** and **Neupy** libraries were used for developing the neural network models. **MAE** was chosen as performance evaluation metric for the regression problems. We made use of **Multi-Layer Feedforward Neural Network with Back Propagation** and **General Regression Neural Network (GRNN).** The results of these regression models were aggregated to give a novel ensemble model for regression on this dataset. An exploratory analysis using unsupervised learning was also performed on this data set using **Self Organizing Maps (SOMs)**, which is available in the **Neupy** library.

1. Classification using Neural networks

1.1 Data Understanding

The **Fashion MNIST** data set is composed of 10 categories of article images of Zalando, a German Ecommerce company. The data set was originally obtained from Kaggle and the link to the data is here. The dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes [2].

The ten categories of images in order are T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot. A label of 0 would mean the image is a T-shirt while a label of 9 would mean that the image is of an ankle boot and so on as shown below:

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boots

A representative image of each of the categories is shown here:

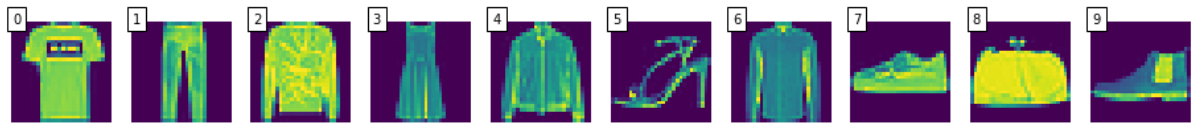


Figure 1 Fashion MNIST - the 10 categories

The objective of the classification is to classify the test images into one of these 10 categories.

1.1.1 Initial data

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

The training data is split into 2 files, where the first file **train-images-idx3-ubyte** contains the 784-pixel values for each of the training images. A sample of the data for the first 5 rows and 10 columns is shown below

Pixel0	Pixel1	Pixel2	Pixel3	Pixel4	Pixel5	Pixel6	Pixel7	Pixel8	Pixel9
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	5	0	0
0	0	0	1	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

The other training data file **train-labels-indx1-ubyte** contains the labels associated with each of the training images. A sample is shown below for the first 5 rows

label
9
0
0
3
0

To locate a pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix.

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top [2].

The same pattern also applies to the test data where the data is split into 2 files **t10k-images-idx3-ubyte** and **t10k-labels-idx1-ubyte** containing the image pixel details and the labels for the test images respectively.

1.1.2 Data exploration

The initial 5 rows of the training data were plotted using **matplotlib** and the following images were observed. This corresponds to the labels for the first 5 training data.

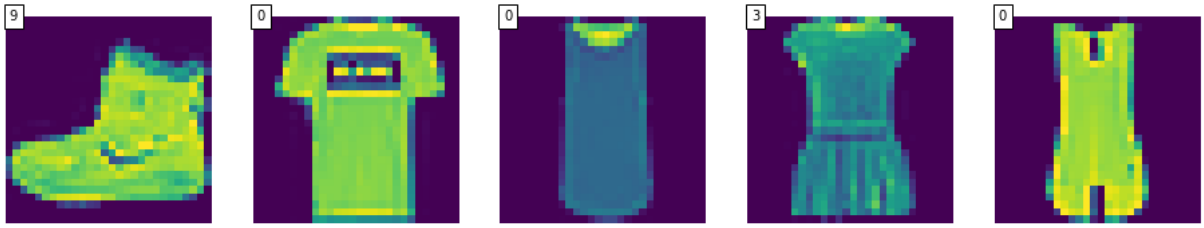


Figure 2 Labels for 1st 5 Training Data

1.1.3 Data quality

There was no missing pixel information or label for any of the training data and so there was no imputation performed.

1.2 Data Preparation

1.2.1 Data selection and pre- processing

The following transformations were applied to the input training images before they were passed as inputs to the different neural networks.

a. Reshape: Each of the training example which had the shape (784) was transformed to the shape (28, 28, 1). It was done since it was known that the height and width of the training images were 28 pixels each. The final 1 in the dimension implied that the images were monochromatic.

b. Convert datatype of pixel intensity to float: The original pixel intensity was an integer between 0 and 255, but to ensure that when the values were divided a decimal value would be returned rather than an integer, they were transformed to floating point numbers.

c. Scaling of pixel intensities: All the pixel intensities which were between 0 and 255 were scaled to a value between 0 and 1 by dividing the intensity with 255.

d. One-hot encoding of the labels: The class labels for the images, i.e. the Y variables were encoded via a one-hot encoding. The one-hot vector included 10 elements, with a 1 denoting that the image belonged to that class. So, an image with label 5 would be classified as 0000010000 since the index of the vector starts from 0.

1.2.2 Partitioning available data for training and testing

The data obtained for the classification exercise was already divided into the training and the testing sets. There were 60,000 images for training and 10,000 images for testing.

All pre-processing that were performed on the training data – scaling pixel values between 0 and 1, one-hot encoding of the output label, etc. were also performed on the test data set.

1.2.3 Data augmentation

The popular image processing library, imgaug [3] was leveraged to augment the training images to generate rich training samples. The augmentation settings can also be viewed as hyperparameters, for example, Gaussian blur of sigma 1 was able to improve the accuracy by 0.2% and a higher value decreased the accuracy, while pixel dropout did not improve accuracy.

1.3 Neural network models and their evaluation

This section will discuss the various neural network architectures that were developed to classify the images. The primary criterion for evaluating the Neural networks was the classification accuracy which is the number of test image classes correctly predicted divided by the total number of test images, which was 10,000.

1.3.1 Multi-layer feedforward Neural network with back propagation (MLFF-BP)

A multi-layer feedforward neural network with back propagation was designed to classify the test images as the first approach. The network had 3 hidden layers.

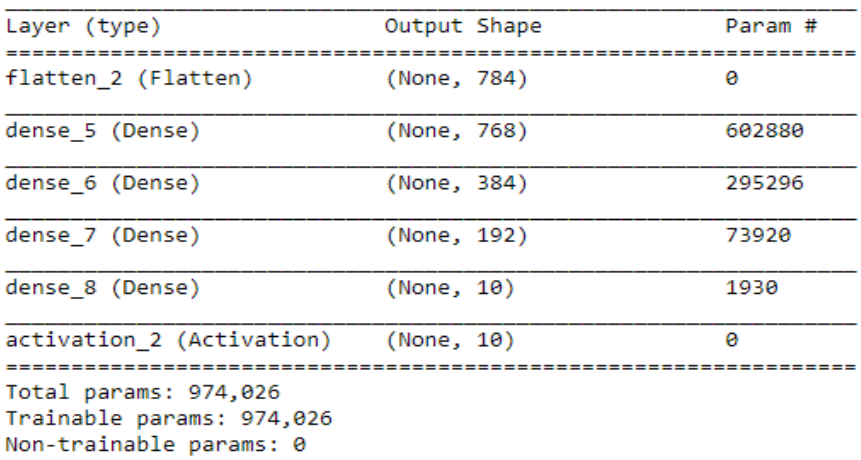
1.3.1.1 Tool used

The **Keras** library with a **TensorFlow** backend was used for developing the Neural network.

1.3.1.2 Architecture

The Neural network had a flattening input layer followed by 4 fully connected layers of different number of output dimensions and finally an output layer with the Softmax activation function for multi-class classification. Each of the dense layer had a ReLU activation function and the weights were initialized by a uniform distribution.

The architecture of the Neural Network was as shown below:



Layer (type)	Output Shape	Param #
flatten_2 (Flatten)	(None, 784)	0
dense_5 (Dense)	(None, 768)	602880
dense_6 (Dense)	(None, 384)	295296
dense_7 (Dense)	(None, 192)	73920
dense_8 (Dense)	(None, 10)	1930
activation_2 (Activation)	(None, 10)	0

Total params: 974,026
Trainable params: 974,026
Non-trainable params: 0

Figure 3 Classification - Multi-Layer Feedforward Neural Network

The categorical cross entropy was chosen as the loss function and Stochastic Gradient Descent (SGD) was chosen as the optimizer to learn the parameters efficiently.

1.3.1.3 Hyperparameters

The hyperparameters for the neural network were the following:

a. Learning rate for the Stochastic Gradient Descent (SGD) = 0.1

b. Momentum for SGD = 0.9

c. Decay for SGD = 0.000001

d. Nesterov momentum was chosen

e. Number of epochs for training = 50

f. Batch size for the SGD = 64

1.3.1.4 Evaluation

When this trained neural network was applied to the test data, an accuracy of **0.884** was obtained. The test data was passed to the model as validation data along with the training data for the evaluation purpose.

The below image shows the categorical cross-entropy loss with the progress of epochs of training on the training and validation data:

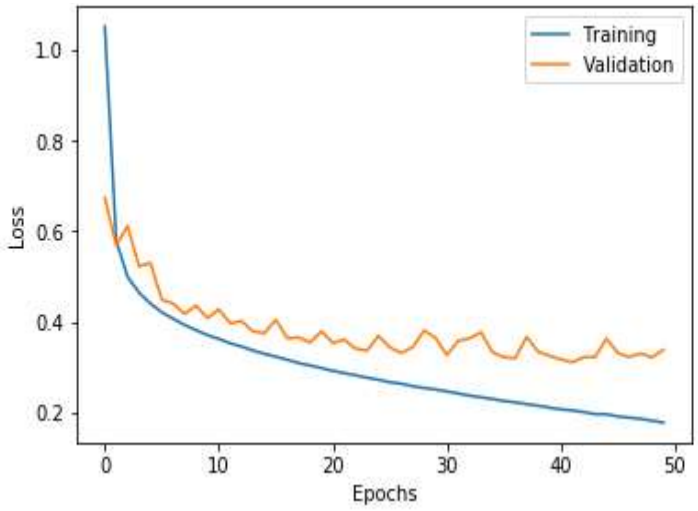


Figure 4 Classification - Categorical Cross-Entropy Loss for MLFF Neural Network

The below image shows the accuracy with the progress of epochs of training on the training and validation set:

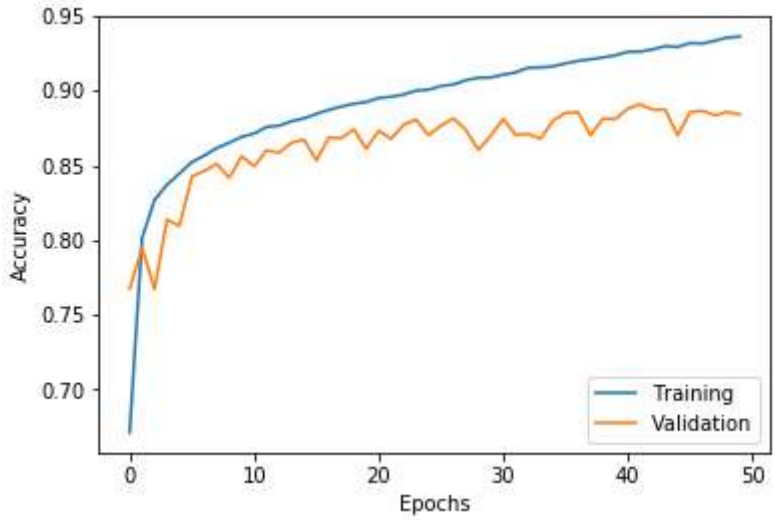


Figure 5 Classification - Accuracy Graph for MLFF Neural Network

Based on the above images, it can be seen that the model's accuracy on the validation data had become consistent by the end of the 50th epoch.

1.3.2 Convolutional Neural Network (CNN)

There were multiple CNN architectures that were evaluated with increasing order of layers. The below describe the architectures of the CNNs that were developed and their performances.

1.3.2.1 Tool used

The **Keras** library with a **TensorFlow** backend was used for developing the Neural network. **Scikit-Learn** and **Matplotlib** was used to generate the graphs and other evaluation metrics.

1.3.2.2 Architecture

We tried using the following CNN architectures to classify the MNIST fashion data:

1.3.2.2.1 CNN with one convolutional layer

This was the first approach of a convolutional neural network with only one convolutional layer. A few activation layers with ReLU were also added and the below architecture was the result

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_5 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 21632)             0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               2769024
_________________________________________________________________
activation_6 (Activation)    (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_7 (Activation)    (None, 10)                0
=================================================================
Total params: 2,770,634
Trainable params: 2,770,634
Non-trainable params: 0
_________________________________________________________________
```

Figure 6 Classification - CNN with One Convolutional Layer

Evaluation

The validation accuracy decreases sharply near 45 epochs, which meant the model had been over-trained slightly when it was trained for 50 epochs.

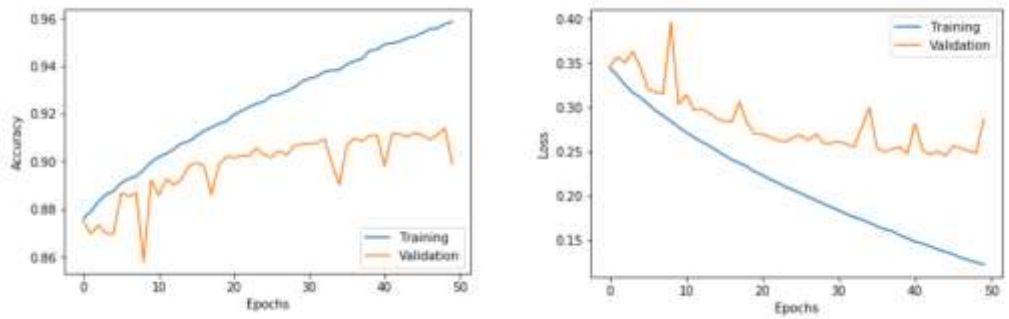


Figure 7 Classification - CNN with 1 Convolutional Layer (Accuracy and Loss)

1.3.2.2.2 CNN with two convolutional layers, max pooling layers and dropout

This was the next approach with 2 convolutional layers, max pooling layers to reduce the number of parameters and finally dropout as a regularization technique to avoid over-fitting.

A summary of the network is provided below:

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 26, 26, 32)	320
activation_8 (Activation)	(None, 26, 26, 32)	0
conv2d_5 (Conv2D)	(None, 24, 24, 32)	9248
activation_9 (Activation)	(None, 24, 24, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 32)	0
dropout_1 (Dropout)	(None, 12, 12, 32)	0
flatten_4 (Flatten)	(None, 4608)	0
dense_5 (Dense)	(None, 128)	589952
activation_10 (Activation)	(None, 128)	0
dropout_2 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 10)	1290
activation_11 (Activation)	(None, 10)	0

Total params: 600,810
Trainable params: 600,810
Non-trainable params: 0

Figure 8 Classification - CNN with 2 Convolutional Layers, Batch Normalization and Dropout

Evaluation

Training was performed for 50 epochs and an accuracy of **0.91** was observed.

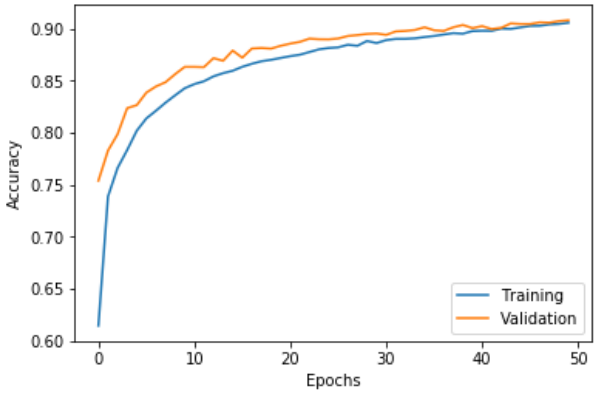


Figure 9 Classification - CNN with 2 Convolutional Layers - Accuracy

1.3.2.2.2 Deep CNN with Multiple Layers, Drop-out, Batch Normalization and Image Augmentation

Adding batch normalisation increases test accuracy to **0.9298**. Doubling the number of convolutional matrix filters improves the test accuracy to **0.9357**. Gaussian blur was used for data augmentation as we found that it contributed most to test accuracy. Here is the summary of the final architecture.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d_5 (MaxPooling2	(None, 13, 13, 64)	0
batch_normalization_9 (Batch	(None, 13, 13, 64)	256
dropout_9 (Dropout)	(None, 13, 13, 64)	0
conv2d_8 (Conv2D)	(None, 11, 11, 128)	73856
max_pooling2d_6 (MaxPooling2	(None, 5, 5, 128)	0
batch_normalization_10 (Batc	(None, 5, 5, 128)	512
dropout_10 (Dropout)	(None, 5, 5, 128)	0
conv2d_9 (Conv2D)	(None, 3, 3, 256)	295168
batch_normalization_11 (Batc	(None, 3, 3, 256)	1024
dropout_11 (Dropout)	(None, 3, 3, 256)	0
flatten_3 (Flatten)	(None, 2304)	0
dense_5 (Dense)	(None, 256)	590080
batch_normalization_12 (Batc	(None, 256)	1024
dropout_12 (Dropout)	(None, 256)	0
dense_6 (Dense)	(None, 10)	2570

Total params: 965,130
Trainable params: 963,722
Non-trainable params: 1,408

Figure 10 Classification - Deep CNN with Multiple Layers, Drop-out, Batch Normalization and Image Augmentation

Evaluation

The first 50 epoch was trained with learning rate 0.001, followed by another 50 epochs with learning rate 0.0001. A test accuracy of **0.9414** was obtained. As there was no improvement after 20 epochs, we used the model at the 20th epoch mark.

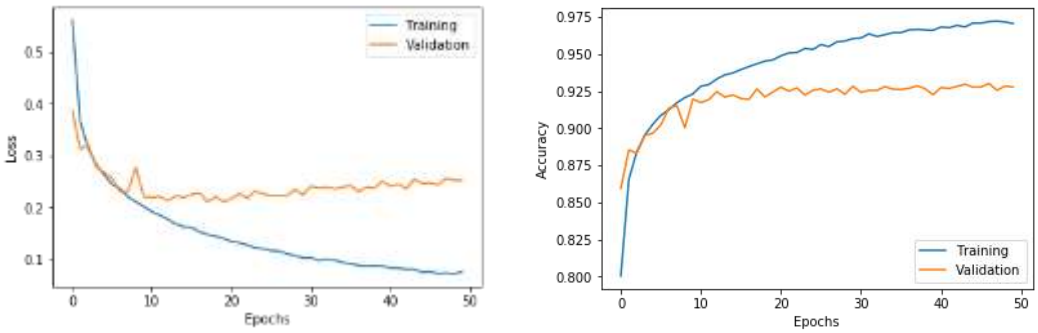


Figure 11 Classification - Deep CNN - Accuracy and Loss

1.3.3 Recurrent Neural Networks (RNN)

1.3.3.1 Tool used

Keras with the **Tensorflow** backend was used to train the Recurrent Neural Networks. Matplotlib was used to plot the various graphs and other evaluation metrics.

1.3.3.2 Architecture

We tried using the following RNN architectures to classify the Fashion-MNIST data.

1.3.3.2.1 Simple Recurrent Neural Network

The data was fed into the RNN with the image's rows as the time steps (28 pixels in total) and the image's columns as the input dimension (also 28 pixels in total). We used 50 as the number of hidden neurons in the recurrent neural network layer. This layer was followed by a dense layer that outputs 10 classes. SoftMax activation function was then used to generate the probabilities of each class.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 50)                3950
_________________________________________________________________
dense_1 (Dense)              (None, 10)                510
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 4,460
Trainable params: 4,460
Non-trainable params: 0
_________________________________________________________________
```

Figure 12 Classification - Simple RNN

The categorical cross entropy was chosen as the loss function and Adam Optimizer was chosen as the optimizer to learn the parameters efficiently. An early stopping criterion was utilized such that if the validation accuracy did not improve for 5 epochs, the training would stop.

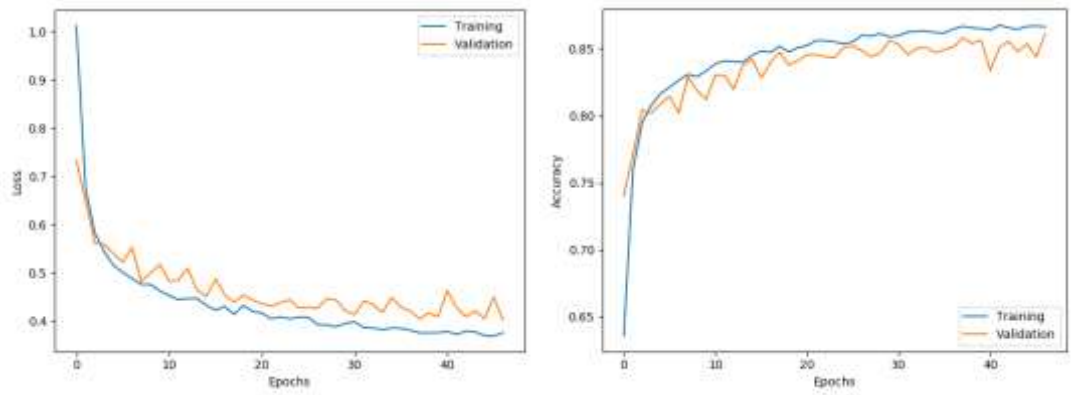


Figure 13 Classification - Simple RNN Accuracy and Loss

Hyperparameters

The hyperparameters for the simple recurrent neural network were the following:

a. Learning rate for Adam Optimizer = 0.001

b. Beta_1 = 0.9

c. Beta_2 = 0.999

d. Epsilon = None

e. Decay = 0.0

f. Amsgrad = False

Evaluation

The early stopping criterion caused the training to stop at epoch number 45. Upon evaluation on the test data, an accuracy of **0.8609** was obtained.

1.3.3.2.2 Multi-Layer LSTM

In this network, we used the LSTM cell, and stacked an additional LSTM layer of 32 neurons. Also, a dropout layer of 20% probability was added in between the two LSTM layers. The rest of the architecture remained similar to that of the Simple RNN.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 28, 50)	15800
dropout_1 (Dropout)	(None, 28, 50)	0
lstm_2 (LSTM)	(None, 32)	10624
dense_4 (Dense)	(None, 10)	330
activation_4 (Activation)	(None, 10)	0

Total params: 26,754
Trainable params: 26,754

Figure 14 Classification - Multi-Layer LSTM

The categorical cross entropy was chosen as the loss function and Adam Optimizer was chosen as the optimizer to learn the parameters efficiently. An early stopping criterion was utilized such that if the validation accuracy did not improve for 5 epochs, the training would stop.

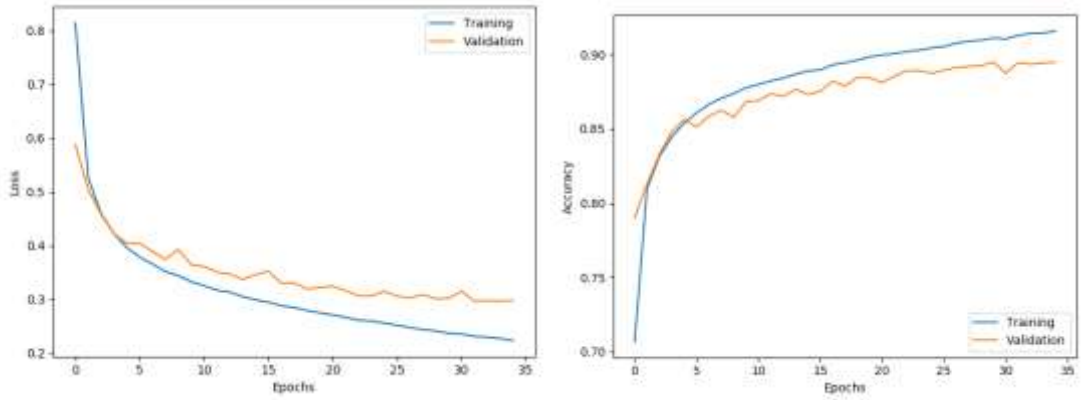


Figure 15 Classification - Multi-Layer LSTM Accuracy and Loss

Hyperparameters

The hyperparameters for the simple recurrent neural network were the following:

a. Learning rate for Adam Optimizer = 0.001

b. Beta_1 = 0.9

c. Beta_2 = 0.999

d. Epsilon = None

e. Decay = 0.0

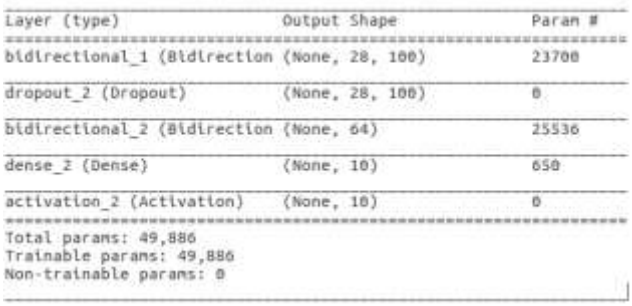
f. Amsgrad = False

Evaluation

The early stopping criterion caused the training to stop at epoch number 34. Upon evaluation on the test data, an accuracy of **0.8949** was obtained.

1.3.3.2.3 Bi-directional GRU

In this model, we used the GRU cell instead of LSTM. We also used a bi-directional wrapper over the GRU cell to create a Bi-directional RNN. Again, a dropout of 20% was introduced, followed by another bi-directional GRU cell. The rest of the architecture remained as in previous examples.



Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirection	(None, 28, 100)	23700
dropout_2 (Dropout)	(None, 28, 100)	0
bidirectional_2 (Bidirection	(None, 64)	25536
dense_2 (Dense)	(None, 10)	650
activation_2 (Activation)	(None, 10)	0

Total params: 49,886
Trainable params: 49,886
Non-trainable params: 0

Figure 16 Classification - Bi-directional GRU

The categorical cross entropy was chosen as the loss function and Adam Optimizer was chosen as the optimizer to learn the parameters efficiently. An early stopping criterion was utilized such that if the validation accuracy did not improve for 5 epochs, the training would stop.

Hyperparameters

The hyperparameters for the simple recurrent neural network were the following:

a. Learning rate for Adam Optimizer = 0.001

b. Beta_1 = 0.9

c. Beta_2 = 0.999

d. Epsilon = None

e. Decay = 0.0

f. Amsgrad = False

Evaluation

The early stopping criteria caused the training to stop at epoch number 36. Upon evaluation on the test data, an accuracy of **0.9021** was obtained. This is the one we chose for the RNN portion of the ensemble.

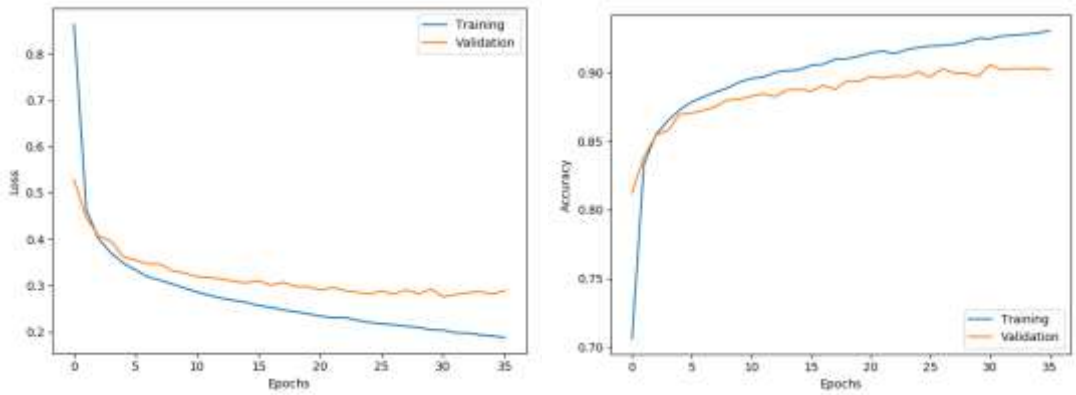


Figure 17 Classification - Bi-directional GRU Accuracy and Loss

1.3.4 Exploratory Model – Capsule Networks

We experimented with Capsule Network [1] and added a dropout layer in decoder. Capsule network was able to produce a test accuracy of **0.9163**.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 28, 28, 1)	0	
conv1 (Conv2D)	(None, 20, 20, 256)	20992	input_1[0][0]
primarycap_conv2d (Conv2D)	(None, 6, 6, 256)	5308672	conv1[0][0]
primarycap_reshape (Reshape)	(None, 1152, 8)	0	primarycap_conv2d[0][0]
primarycap_squash (Lambda)	(None, 1152, 8)	0	primarycap_reshape[0][0]
digitcaps (CapsuleLayer)	(None, 10, 16)	1474560	primarycap_squash[0][0]
input_2 (InputLayer)	(None, 10)	0	
mask_1 (Mask)	(None, 160)	0	digitcaps[0][0] input_2[0][0]
capsnet (Length)	(None, 10)	0	digitcaps[0][0]
decoder (Sequential)	(None, 28, 28, 1)	1411344	mask_1[0][0]

Total params: 8,215,568
Trainable params: 8,215,568
Non-trainable params: 0

Figure 18 Classification - Exploratory Capsule Networks

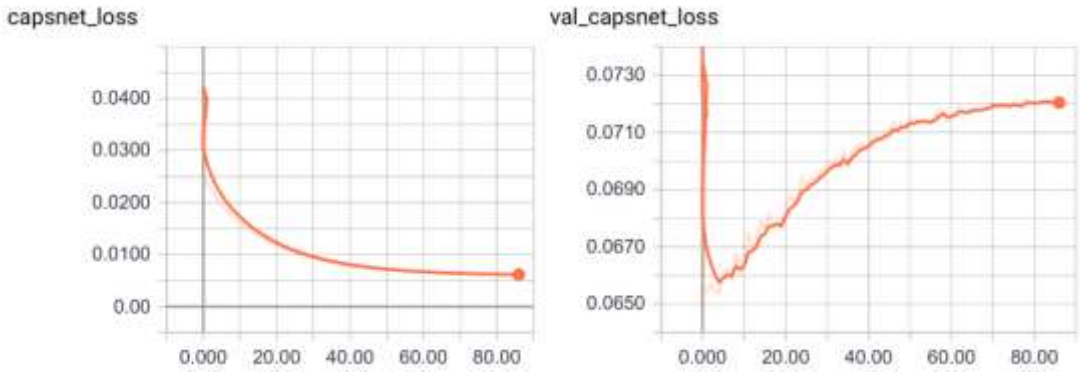


Figure 19 Classification - Capsule Networks Loss for Training and Validation with respect to epochs

Hyperparameters

i. Learning rate = 0.0001

ii. Dropout = 0.2

1.3.5 Comparison of all the algorithms

The below table compares the hyper parameters for the **best architectures of the different Neural network algorithms** that were developed to classify the fashion MNIST images:

Table 1 Different Classification Models Used in Ensemble

Neural network type	**MLFF-BP**	**CNN**	**RNN**
Architecture	4 hidden layers with softmax activation in final layer	Convolutional layers with batch norm and dropout	Bi-directional, multi-layer GRU cell with dropout
Optimizing algorithm and parameters	Stochastic Gradient Descent, Momentum 0.9, Nesterov	Adam, Beta1 0.9, Beta2 0.999	Adam, Beta1 0.9, Beta2 0.999,
Learning Rate	0.1	0.001, 0.0001	0.001
Learning Rate Decay	0.000001	0	0
Batch Size	64	256	256
Epoch	50	50 per learning rate	Early stopping
Prediction Accuracy achieved on Test Set	0.884	0.9414	0.9021

1.3.6 Ensemble Approach to Neural Network Models for Classification

1.3.5.1 Approach

The models from the best CNN, RNN and the MLFF BP neural networks were ensembled using the below formula. Note that we did not include the exploratory capsule network architecture in our ensemble:

Equation 1 Classification Ensemble Approach

$$OutputLabel(i) = argmax(\sum_{m} Accuracy(m) \times \overline{\mathrm{SoftmaxOutputScore(m,\ i)}})$$

Where,

a. **OutputLabel(i)** is the output label between 0-9 associated with the test image, i.
b. **argmax** returns the index of the element in a vector with the highest value. It will return a value between 0 and 9.
c. The summation is taken over all the models in the ensemble.
d. **Accuracy(m)** is the accuracy of the model m from the ensemble. It is a value between 0 and 100.

e. **SoftmaxOutputScore(m, i)** is the output scores of the Softmax function in the last layer in the Neural Network for the model m and the test image i. This is a vector of 10 elements.

The classification accuracy of the ensemble is calculated as the ratio of the number of output labels correctly classified by the ensemble to the total number of test images for classification.

1.3.5.2 Observations

The accuracy of the ensemble was observed to be **0.9434** which is a slight improvement over the best single model test accuracy of **0.9414**.

1.3.5.3 Confusion matrix for ensemble predictions

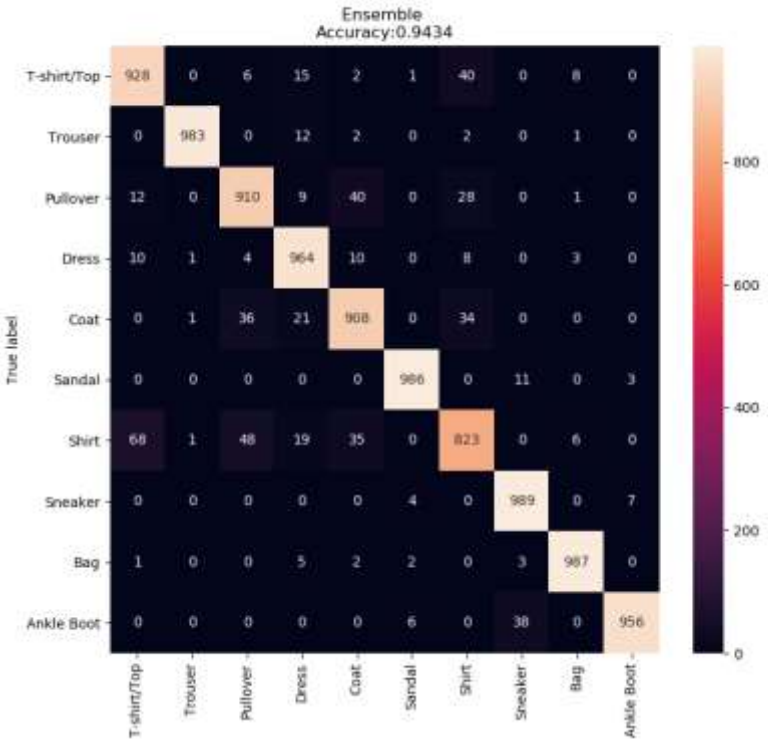


Figure 20 Classification - Ensemble Confusion Matrix

It can be seen in the confusion matrix that most of the mis-predictions happened for articles that resemble each other visually. For example, for the misclassification in the first row, 40 Shirts from the test data set were misclassified as T-shirt/top. Also, the highest number of mispredictions occurred when 68 T-shirts were mis-classified as shirts. We can verify the performance of the shirt in the precision recall table below. This may suggest that more training examples with T-shirts labelled as T-shirts and shirts labelled as shirts could be added to the dataset to allow us to train an even better model.

	precision	recall	f1-score	support
T-shirt/Top	0.91	0.93	0.92	1000
Trouser	1.00	0.98	0.99	1000
Pullover	0.91	0.91	0.91	1000
Dress	0.92	0.96	0.94	1000
Coat	0.91	0.91	0.91	1000
Sandal	0.99	0.99	0.99	1000
Shirt	0.88	0.82	0.85	1000
Sneaker	0.95	0.99	0.97	1000
Bag	0.98	0.99	0.98	1000
Ankle Boot	0.99	0.96	0.97	1000
avg / total	0.94	0.94	0.94	10000

Figure 21 Classification - Ensemble Precision and Recall Values

2. Regression using Neural networks

2.1 Data Understanding

With the expansion of the Internet and Web 2.0, there has also been a growing interest in online news, which allow an easy and fast spread of information around the globe. Thus, predicting the popularity of online news is becoming a recent research trend. Popularity is often measured by considering the number of interactions in the Web and social networks (e.g., number of shares, likes and comments). Predicting such popularity is valuable for authors, content providers, advertisers and even activists/politicians (e.g., to understand or influence public opinion). The data has been obtained from UCI website and link is here. This dataset summarizes a heterogeneous set of features about articles published by Mashable in a period of two years. The goal is to predict the number of shares of the article in social networks (popularity) [4].

2.1.1 Initial data

The online news popularity dataset is in **CSV** format and it consists of **39,644** records with **61** attributes. Among 61 attributes, **58** are predictive attributes, **2** are Non-Predictive (url, timedelta) and **1** is target attribute (Shares).

2.1.2 Data description

The below table summarizes the original data types and what they represented, which was obtained from different sources [5] and [4]:

Table 2 Attributes of News Popularity Dataset

#	Column name	Column description	Data type
1	url	The URL of the article in Mashable	String
2	timedelta	Days between the article publication and the dataset acquisition	Real
3	n_tokens_title	Number of words in the title	Real
4	n_tokens_content	Number of words in the article	Real

5	n_unique_tokens	Rate of unique words	Real number between 0 and 1
6	n_non_stop_words	Rate of non-stop [1]words	Real number between 0 and 1
7	n_non_stop_unique_tokens	Rate of unique non-stop words	Real number between 0 and 1
8	num_hrefs	Number of links	Real
9	num_self_hrefs	Number of Mashable article links	Real
10	num_imgs	Number of images in the article	Real
11	num_videos	Number of videos in the article	Real
12	average_token_length	Average length of the words in the content	Real
13	num_keywords	Number of keywords	Real
14	data_channel_is_X	The channel in which the article appeared. The value of X is one from lifestyle, entertainment, business, social media, technology, world. Based on the available data, an article can have only one channel and so the value for that channel is a 1.	Bool
15	kw_X_Y	X and Y are one of min, avg and max. They denote the minimum, average and maximum shares for the worst (min), average (avg.) and best (max) keywords that occur in the article.	Real
16	self_reference_X_shares	X is one of min, max and avg. It represents the X shares of referenced articles in Mashable.	Real
17	Weekday_is_X	X is one of the seven days of the week. Based on the available data only one of these group of columns has a true value for an article.	Bool
18	LDA_X	X is one from 0 through 4. It shows the similarity of the content in the article to the top 5 LDA[2] topics.	Real number between 0 and 1
19	global_subjectivity	Article text subjectivity score	Real number between 0 and 1
20	global_sentiment_polarity	Article text polarity score	Real number between 0 and 1

[1] A stop word is a word like “the” which represent the commonly used words and are typically ignored in Natural Language Processing (NLP) tasks

[2] LDA is a NLP algorithm that helps identify the top relevant topic among all the available Mashable texts and then measure the closeness of the article with that topic

21	global_rate_positive_words	Rate of positive words in the content	Real number between 0 and 1
22	global_rate_negative_words	Rate of negative words in the content	Real number between 0 and 1
23	rate_positive_words	Rate of positive words among non-neutral tokens	Real number between 0 and 1
24	rate_negative_words	Rate of negative words among non-neutral tokens	Real number between 0 and 1
25	X_positive_polarity	X is one of min, avg and max. It represents the X polarity of positive words.	Real number between 0 and 1
26	X_negative_polarity	X is one of min, avg and max. It represents the X polarity of negative words.	Real number between -1 and 0
27	title_subjectivity	Article title subjectivity measure	Real number between 0 and 1
28	title_sentiment_polarity	Title polarity	Real number between -1 and 1
29	abs_title_subjectivity	Absolute subjectivity level	Real number between 0 and 1
30	abs_title_sentiment_polarity	Absolute polarity level	Real number between 0 and 1
31	shares	The number of shares for the Mashable article after the value has stabilized. This is the **target** for the regression analysis.	Real

2.1.3 Data exploration

An initial analysis of the data was done to get familiar with the training data. The below are some of the important observations.

2.1.3.1 Correlation analysis

The below image shows the correlation analysis between the original variables

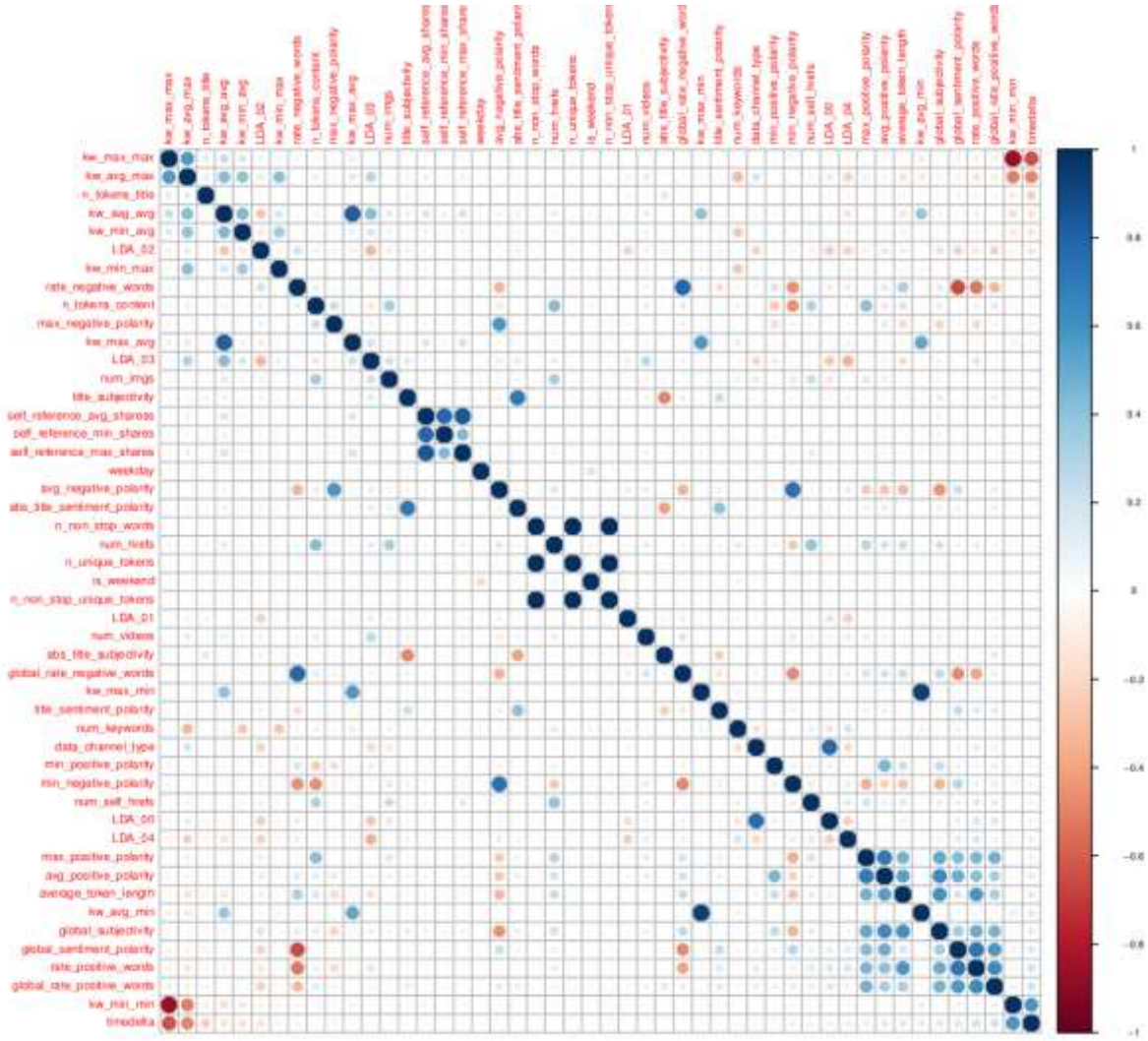


Figure 22 Regression - Correlation Matrix

2.1.3.2 Distribution of shares (target variable)

The below graph and table shows the distribution of the shares on normal quartile plot, is in the form of an exponential function. This contributed to choosing MAE as the evaluation metric for the regression models.

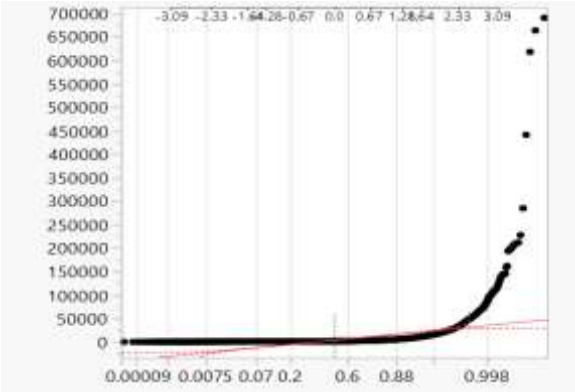


Figure 23 Distribution of the Shares Target Attribute

2.1.3.3 Day of week vs shares

The number of shares, which is the target variable was plotted with the days of the week and the below plot was obtained for the training data

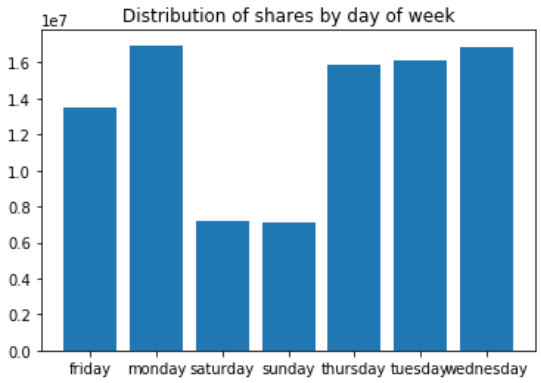


Figure 24 Distribution of Shares by Day of Week

It can be seen from the above figure that the number of shares are especially low for Saturday and Sunday, which are weekends.

2.1.3.4 Channel vs shares

The number of shares was also plotted against the channel in which the article was posted and the below plot was obtained:

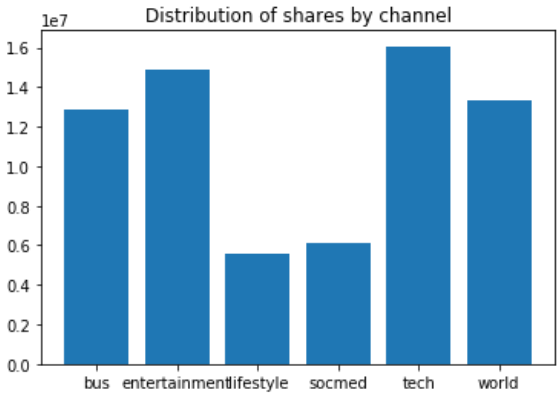


Figure 25 Distribution of shares by channel

It can be seen from the above figure that the number of shares are especially low for lifestyle and social media channels.

2.2 Data Preparation

2.2.1 Removal of one-hot encoded channel

The columns that represented the one-hot encoded channels for the article were removed and a single column called **is_life_style_social_media** was used to replace them since it was found during the data exploration that the number of shares significantly varied for life style or social media channel compared to the other channels.

2.2.2 Removal of one-hot encoded weekday

The columns that represented the one hot encoded day of the week when the article was published were also removed. This was done since it was found that the number of shares varies considerably between weekdays and weekends and the existing column **is_weekend** already captured whether the article was published on a weekend or not.

2.2.3 Removal of aggregate score values

Some of the columns carried aggregate metrics like min, max and average of the particular feature. The min and max values were removed and only the average value features were kept. This was done since it was felt that the average values represented the underlying feature the best and removing the extra columns would make the training process faster. Also, some of these aggregate columns had high correlation among themselves since a record that had a higher minimum for the feature would also have a higher maximum.

2.2.4 Removal of time delta variable

It was stated that the data was collected after the sharing for the article had stabilized i.e. there were no longer any more shares for some time. In this respect, the "timedelta" variable would be not considered as an important factor.

2.2.5 Removal of other identity columns and constant columns

The URL column was removed as it was an identity column.

2.2.6 Conversion to one column for LDA data

As described above there are 5 columns that show how similar the topic in the article is to one of the 5 LDA topics. Only the maximum LDA value was kept as a measure of the closest the article is to any one of the 5 LDA topics. The new column was named **max_closeness**.

2.2.7 Normalization of input features

All the input features in the training set and the target variable (number of shares) were normalized since their ranges varied a lot. This was done to speed up the learning process [6]. A few different normalization methods were tried to get better MAE scores. The following classes from the Scikit library were evaluated for normalizing the data: MinMaxScaler, StandardScaler and RobustScaler. We found that MinMaxScaler gave the least error among all the normalizers.

Normalization of the input features and the actual value of the target variable in the validation set were also carried out using the same parameters of the chosen normalizer obtained from the training set.

2.2.8 Partitioning data for training and testing

The original data has been divided as **70%** training and **30%** testing using random sampling using the Python Scikit learn library methods. Thus, the training dataset contains **27,750** records and the testing dataset contains **11,894** records.

2.3 Metric chosen to evaluate network performance – MAE vs. RMSE

The Mean Average Error or MAE was chosen as the metric to evaluate the different neural networks that were developed for the regression analysis. The MAE was chosen over the Root Mean Square Error (RMSE) since it is easier to interpret the MAE compared to the RMSE [7]. This would mean that from the MAE value for the shares we can get the absolute error in the number of shares by inverse scaling the normalized number of shares.

Also, in our analysis we would like to find out which algorithm has predictions which are on average closer to actual number of shares and it would be of no benefit to penalize higher differences between the predicted and the actual values more, which is done by the RMSE measure. The target variable follows an exponential distribution as seen during data analysis so a few predictions that might be widely different from the actual values would contribute excessively to the RMSE measure.

Finally, the MAE is easier to compute, since for each validation vector, only the absolute difference between the predicted and actual value needs to be computed, which would help to ease the implementation required to evaluate the performance of the ensemble regression model.

2.4 Neural network models and their evaluation

The following sections describe the various neural network models that were developed for the regression analysis and evaluates the same.

2.4.1 Multi-Layer Feedforward Neural Network with Back Propagation

2.4.1.1 Tool used

We used **Scikit-Learn** for the Multi-Layer Feedforward with Back Propagation due to its flexibility with randomized search and early stopping. Matplotlib was used to plot the various graphs and other evaluation metrics.

2.4.1.2 Hyperparameters

After doing a randomized search with 3-fold cross validation, we eventually settled upon the following parameters.

a. Learning Rate Mode = constant

b. Initial Learning Rate = 0.0218

c. Solver = SGD

d. Activation Function = tanh

e. Alpha = 0.215

f. Beta_1 = 0.5

g. Beta_2 = 0.76

2.4.1.3 Architecture

We used a 3-layer neural network and employed the use of early stopping to prevent over-training the model. We used the default early-stopping scheme provided by Scikit-Learn, which states that MLPRegressor (Multi-Layer Perceptron Regressor) would set aside 10% of the training data as validation data, and terminate training when validation score is not improving by the tolerance for the optimization (tol) parameter (default value: 1e-4) for at least two epochs.

2.4.1.4 Evaluation

We got a **MAE value of 0.0043** for the test data set. The MAE value of 0.0043 translates to an average error of **2971** for the predicted shares.

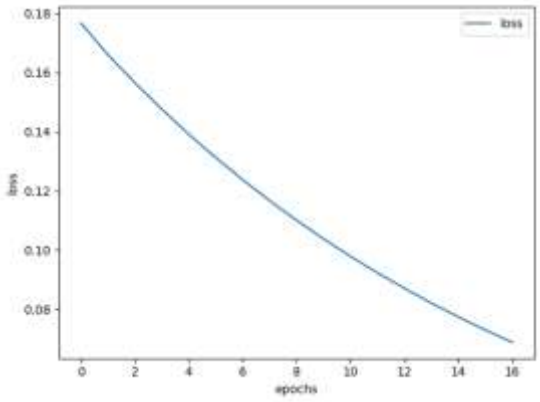


Figure 26 Regression - Loss Curve for MLFF

2.4.2 General Regression Neural Network (GRNN)

2.4.2.1 Tool used

We used Neupy for the GRNN model as it had a robust GRNN implementation.

2.4.2.2 Hyperparameters

The only hyperparameter available to the GRNN function in Neupy was the standard deviation. A grid-search approach was taken to evaluate the MAE of the GRNN against different values of the standard deviation to find the optimal value for standard deviation.

2.4.2.3 Architecture

The standard GRNN architecture with input neurons, pattern neurons, summation neurons and an output neuron was used.

2.4.2.4 Evaluation

The below graph shows the variation of MAE with the standard deviation parameter of the neural network.

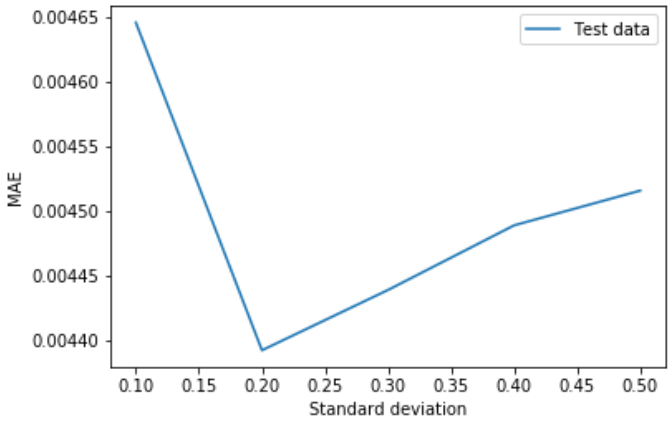


Figure 27 Regression - MAE Values for Different Std. Dev. Values

As can be seen from the above graph, the least MAE was observed when the standard deviation was very close to 0.2. The value of MAE for a standard deviation of 0.2 was **0.0044**.

2.4.3 Regression with an ensemble of neural networks

2.4.3.1 Approach

The results from the best MLFF BP regressor model and the General Regression Neural Network (GRNN) were ensembled using the below formula:

Equation 2 Regression - Calculation of Predicted Shares for Ensemble Model

$$PredictedSharesForEnsemble(\vec{t}) = \sum_{m} \frac{\frac{1}{(1+MAE(m))}}{\beta} \times PredictedShares(m, \vec{t})$$

$$\beta = \sum_{m} \frac{1}{(1 + MAE(m))}$$

Where,

a. **PredictedSharesForEnsemble(t)** is the predicted number of shares by the ensemble for the validation vector t
b. The summation is taken over all the models in the ensemble – Multi layer feedforward neural network with back propagation and GRNN
c. **MAE(m)** is the Mean Absolute Error that was derived for the model m during the validation stage
d. **PredictedShares(m, t)** is the predicted number of shares (target variable's value) for the model m on the validation vector t

The idea of the ensemble is to give a higher weightage to the regression model with lower Mean Absolute Error. The ensemble does a calibration of the weights of the predicted shares for the individual models to arrive at the predicted shares for the ensemble.

The 1 added to the denominators ensures that the value of the fraction is defined even when the MAE approaches 0 for any model in the ensemble. Also, for values of MAE very close to 0, the value of the fraction will not be so huge that it suppresses the predictions from the other models.

The MAE for the Ensemble would be calculated as the mean of the absolute errors for each of the validation records that are predicted by the ensemble.

2.4.3.2 Observations

The combined MAE of the ensemble was found to be **0.0040** and this translated roughly to an MAE in terms of number of shares of **2753**.

3. Unsupervised learning using Self-organizing maps (SOM)

3.1 Self-Organizing Maps

Self-Organizing Maps were explored on this regression data set to divide the training data into clusters and see if any useful insights could be gathered.

3.1.1 Tool used

NeuPy was used, which is a Python library for artificial neural networks. For the SOM algorithm, the API documented here was used.

3.1.2 Approach

The training data after the processing described earlier was fed to Scikit-Learn's PCA algorithm and 2 principal components were created from the training data. The Principal component values were fed as

input to the SOM to get the cluster centres. The number of output clusters were varied between 3, 4 and 5 and the clusters were analysed based on the positions of the final cluster centres.

3.1.3 Hyperparameters

The learning rate or step was kept at 0.1.

The learning radius was 0 to ensure that the clusters were independent of each other which is required in a clustering solution.

The grid type for the network was kept a rectangle.

The weights of the neurons in the network was initialized using a normal distribution.

The distances were calculated using the Euclidean distance metric.

The number of epochs over which the network was trained was 100.

3.1.4 Evaluation

The position of the cluster centres for different number of clusters were as shown below:

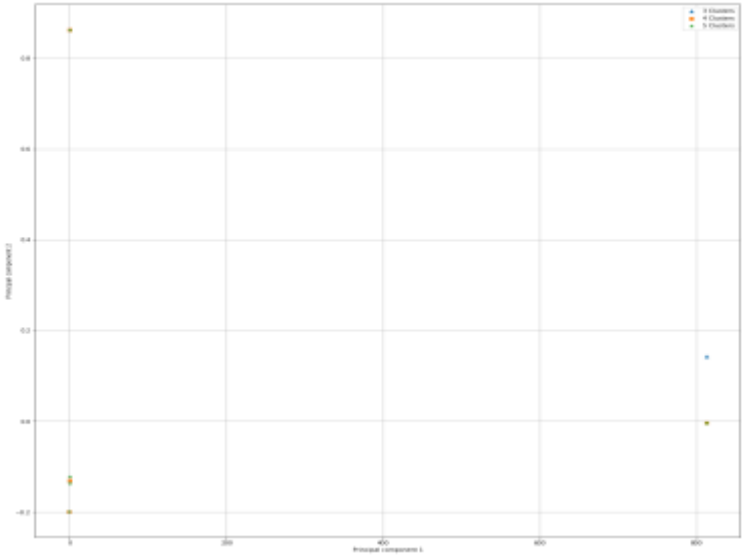


Figure 28 Plot of Principal Component 1 against 2 to see the Clustering (please zoom to see the points)

The data was plotted for the different number of cluster centres as shown below

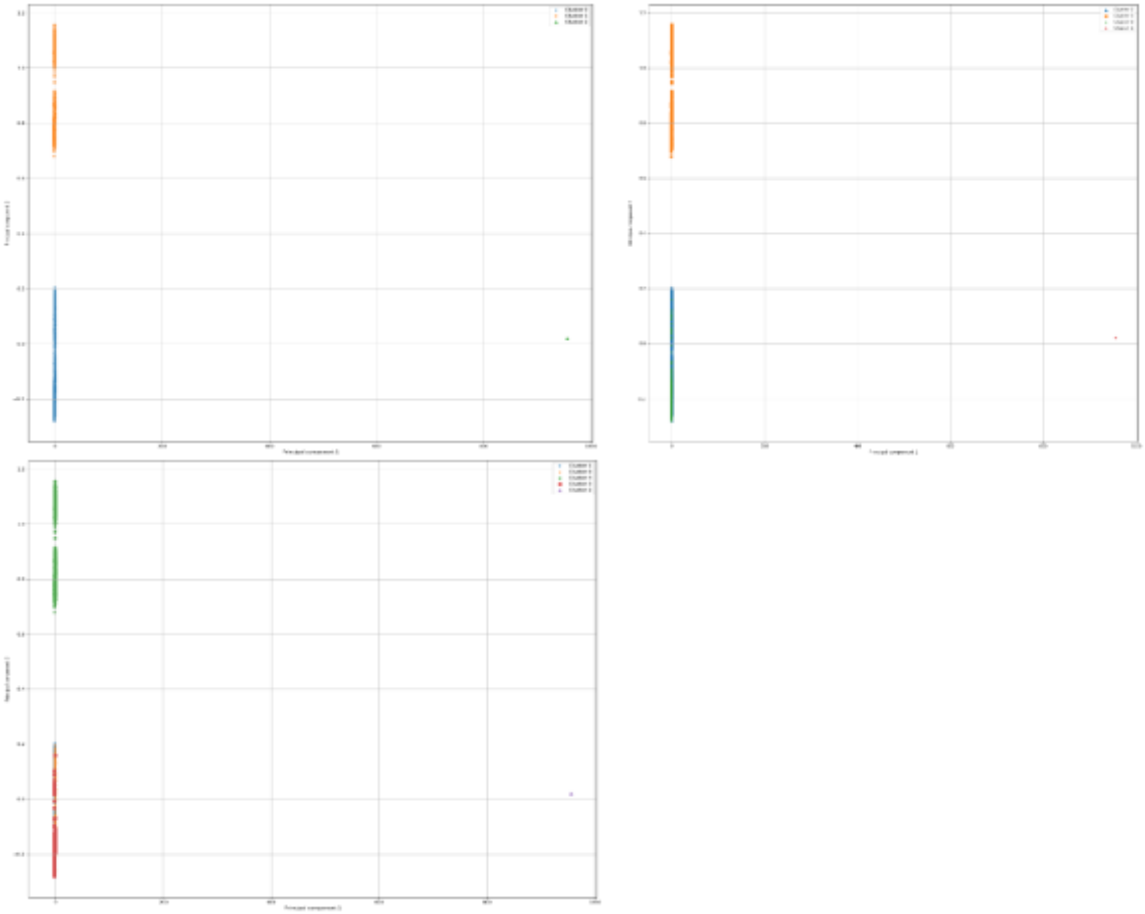


Figure 29 Plot of principal components of training data for number of clusters = 3, 4 and 5

It was seen that the training data could be split into **3** clusters with the best inter cluster distance as can be seen from the first figure above.

4. Conclusion

We employed various neural network algorithms to solve one classification and one regression problem. For the classification problem, we used the MNIST Fashion data set with 10 output classes. The baseline approach with a multi-layer feedforward neural network gave a decent accuracy. This was improved by using Convolutional Neural networks which are specifically designed for image classification tasks. A recurrent neural network was also developed and its performance was compared with the other networks. Finally, we explored, Capsule networks, with good results. These classifiers were then ensembled and the performance of the ensembled classifier was found to be slightly better than that of the standalone models. An accuracy of **0.9434** was observed when using the ensemble of classifiers which was better than all the individual classifiers.

The regression task in this case was to predict the number of shares of a Mashable article in social media platform. The MAE metric was chosen to be minimised as part of the model evaluation phase. To perform the regression, a multi-layer feed forward neural network with a regression output was tried as the baseline approach. A GRNN network was then evaluated whose hyper parameters were tuned to get the optimal MAE. An ensemble of regressors was then created by weighing the different regression models by the inverse of their MAE values and the predicted values were obtained for the ensemble. It was found that the MAE of the ensemble of regressors was slightly better than that of the best regressor that was evaluated.

Finally, as an exploration endeavour, unsupervised learning was performed on this dataset using self-organizing maps. This unsupervised learning was evaluated with different number of clusters. It was found that for **3** clusters the inter cluster separation among the clusters was best captured.

4.1 Lessons learnt

We realised that to get good results from the image classification models, there was a need to leverage GPU hardware. Techniques like early stopping are essential when training to avoid over-training the models. This also saves on power required to train the deep neural networks. Arguably, in order to obtain the optimal hyper-parameters for the neural networks, the methodologies are still not established and require trial and error. Hence, a powerful machine is recommended as this greatly saves on time spent on training the models for evaluation. Leveraging on training that others have done using transfer learning is a good way to stream-line the training process, and would be something that we would attempt if the focus is on accuracy.

For the regression modules, we realised that data understanding and transformation is still important to get good results, even if we are feeding the data into a neural network. Moreover, it is good practice to do proper data understanding and transformation, regardless of the machine learning models we use.

Fortunately, due to concepts learnt during the course most of our efforts were directed to fine tune the model specific hyper parameters.

References

[1] S. S. N. F. Geoffrey E. Hinton, “Dynamic Routing Between Capsules,” 2017.

[2] Zalando Research, “Fashion MNIST Kaggle,” [Online]. Available: https://www.kaggle.com/zalando-research/fashionmnist.

[3] A. Jung, “imgaug Github,” [Online]. Available: https://github.com/aleju/imgaug.

[4] U. M. l. repository, “Online News Popularity dataset,” UCI, 2015. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/online+news+popularity. [Accessed 2018].

[5] P. V. a. P. C. K. Fernandes, “A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News.,” in *17th EPIA 2015*, 2015.

[6] A. Ng, “Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization,” Coursera, [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/lXv6U/normalizing-inputs. [Accessed 2018].

[7] K. M. Cort J. Willmott, “Advantages of the mean absolute error (MAE) overthe root mean square error (RMSE) in assessingaverage model performance,” *Climate Research,* vol. 30, no. 79-82, 2005.

[8] Keras, “Keras,” [Online]. Available: https://keras.io/.

[9] “Online News popularity dataset,” 2015. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/online+news+popularity.