Student Number: 230907

This report details the approach used in classifying whether an image is either memorable or non-memorable, using the given training and test data. The methods utilized will then be discussed with subsequent results analyzed.

1.Approach:

Determining whether an image is memorable or non-memorable could be solved with binary classification. However, given the added dimension of confidence scores we can use a mixture of binary classification and regression approaches. For this task, the multi-layer perceptron regressor (MLP) has been utilized, which is a regressive MLP without an activation layer (which would have given be a binary output) and gives an output on a variable scale of -1 to 1 in this instance. This approach allows us to determine an overall confidence rating for each prediction. We can then convert positive numbers to 1 and negative numbers to 0 to replicate the binary predictions. Beforehand, the training set needs pre-processing as it contains CNN & Gist features on different scales, 'nan' values in 'training2' and high dimensionality with 4608 features per image.

2.Method

The model chosen was the multi-layer perceptron regressor (MLPR). The training confidence and label scores were combined, with negative confidence asserted to 0 labels and positive confidence to 1. This regression model was chosen over standard binary classification as it allows for tuning of multiple hyper-parameters and does not present binary outputs meaning the confidence rating can be utilized on a varying scale of -1 to 1.

A KNN classifier was also used to benchmark the accuracy and confusion matrix results in comparison to the MLPR and to test that the training data was working correctly during pre-processing stages. After label values between of -1 to 1 were outputted from the MLPR, these were converted to 0 or 1, dependent on whether the value was positive or negative.

Normalization was performed using SK learn MinMaxScaler, scaling the data to values of 0-1. This is done to avoid any biased coefficients that may hinder the performance of both the MLPR and the PCA. As the CNN and Gist features are independent, it was decided that normalization would be performed on these features separately. The features will then be unified, with the importance of the features being judged by the PCA.

KNN Imputation (K-nearest neighbors Imputer) is used to fill the MCAR 'nan' features. KNN imputation gives us a multivariate approach to filling in missing features as opposed to the single variate of collecting the mean/median from a given column. The KNN Imputer should thus yield more accurate values. To test the accuracy, the training2 data did not contain the ground truth needed for the missing 'nan' values. To combat this, the approach used, was to remove features from training1 and store the results in a new data set, named training3. The imputation is then used upon the training3 data. The imputed values in training 3 are then compared with the ground truth values from training1. The mean distance between the two values are then converted to a percentage. By multiplying the confidence rating by this new accuracy rating percentage, we can adapt our confidence labels. For the best performance, the KNN imputer, was tuned by testing the number of K-neighbors, increasing/decreasing the training size and by changing the training to validation split ratio. It was also found that by having over 800 samples, alongside tuned hyper-parameters, that the performance gain of the KNN imputer was negligible past this size.

PCA is used in order to reduce the dimensionality from 4608 features to 380 features to within 95% variance. This is used to reduce the quantity of features, reduce noise, whilst minimalizing loss of features that have significant importance for classification. PCA was also used for feature selection, determining which Gist and CNN features have the most significance to our model.

The same normalization, imputation and dimensionality reduction models were re-used on the test data in order to make sure that both training and test data correlated. From the unified training data, validation and training sets were created for use in training the MLPR model. Different splits were tested with an 70/30 being deemed optimum. Different sized training data was also trialed alongside altering the hyperparameters of random seed, learning rate, number of hidden layers and number of nodes within those hidden layers.

3.Results

The highest accuracy score of the validation set for the MLPR was 75.33% which is the same model which has been used to create the predictions. The parameters for this were: 70-30 training to validation split, training size of 1400. Parameters of the MLPR are: 5 hidden layers, nodes quantity: 80,40,40,100,20, 5 random states and a learning rate of 0.01. Please see figure 1. For a visual representation of the accuracy.

Firstly, changing the size of our training data can have a significant impact upon the performance of the KNN imputer and moderately affect accuracy for the MLPR. Figure 2 shows KNN imputer accuracy begins to plateau at 1400 training samples (800 of which are training2). The reason for this is likely to be that below 800, there are not

enough features to compare. The number of K nearest neighbors can also impact accuracy, however by decreasing the number of neighbors we risk overfitting. In terms of the training data size in relation to MLPR accuracy, the graph in figure 2 shows that the accuracy is highest at the size of 1500. From this point onwards, accuracy decreases which could be caused by underfitting.

Hyperparameters were changed and tuned with results shown in figure 4. Accuracy appears to be greatest with a Hidden layer size of 5, however the graph does not show any clear trends. There is also no clear correlation between overall quantity of nodes and accuracy. As these results are limited, I resorted to trial an error in terms of tuning the hidden layers for my MLPR. Deviating from a learning rate of 0.01 and random seeds of 5 also reduces accuracy. Considering the size of the test data, different batch sizes were also trialed without success.

By using an MLP with a binary output, I was able to test against the MLPR, with the same parameters, whether confidence was worth considering. As shown in figure 3, (5 runs of different seeds) the MLP without confidence ratings on average returned higher accuracy. Although against the validation data the accuracy is better, this may not be the case for test cases. By ignoring confidence, we ignore the chance that labels on the validation set with a 66% confidence have a likelihood of being wrong approximately 33% of the time. For this reason, confidence is considered, sacrificing validation accuracy for the possibility of greater test accuracy.

To test the usefulness of the incomplete data, firstly the imputed data was tested. To do this, values were removed from training 1, the values were then filled by the KNNImputer and the accuracy of the values was tested against the original values. This percentage, divided by the number of nan values is then taken from the confidence score. As figure 2 shows, as the training size increases past 1400 the accuracy is past 99% which means the accuracy in terms of imputation should be negligible in comparison to training1. However, this has only been tested against one set of validation data meaning that the imputer could work less favorably against an unseen test set.

Regarding the performance of the PCA, as tested on the KNN classifier, lower dimensionality has cost accuracy. As the PCA is run off overall variance, rather than taking in to account the output of the labels, it is possible the PCA could have removed low variance features that could have an impact on our output label. In hindsight, this may have been better solved with Logistic Regression which would have taken the target label into account. PCA was also fitted to training1 as this data shown to be more accurate. However, training2 is a much larger sample and may have produced better performance in comparison.

Overall, the results from the validation set are promising however far from optimum. Although I have used confusion matrices as well as accuracy for evaluating my model, I could have put more relevance on f1 score as a measurement of performance over accuracy which may have yielded a better evaluation. By using K folds validation to split data for the MLPR, this could have also increased the accuracy of my predictions. It could have been more optimum, to also use a linear regression model followed by a MLP, rather than using the regressor and manually converting the output to binary. One lesson that has been learnt from this experiment is the impact pre-processing can have on the overall accuracy of a classifier.

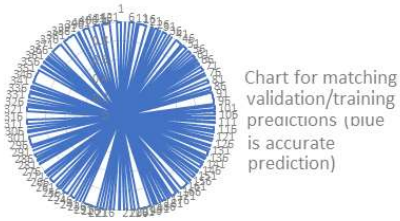

Figure 1: Pie Chart comparing validation/training predictions

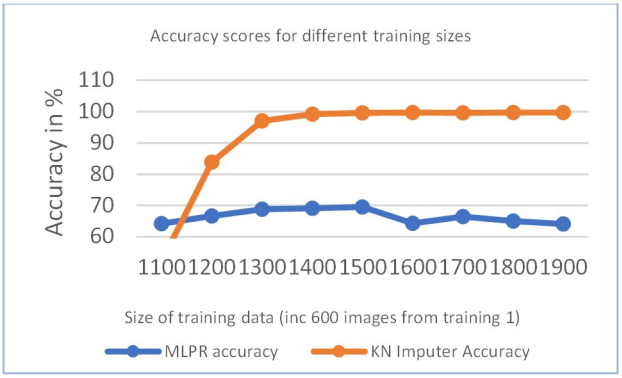

Figure 2: Accuracy for MLPR and KNN Imputer

With/Without Conf	Seed1	Seed2	Seed3	Seed4	Seed5	Average
MLP without confidence	69.73	75.02	70.32	69.69	72.4	71.43
MLP with confidence	70.01	68.52	66.66	70.89	69.83	69.18

Figure 3: accuracy results with and without confidence

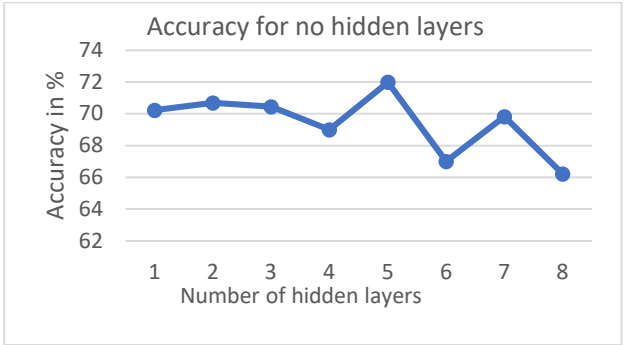

Figure 4: Hidden Layer accuracy (same number of nodes of equal value)

4.References

- 'sklearn.neural_network.MLPRegressor' Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html, Accessed on:06/05/22

- 'sklearn.neural_network models (supervised)' Available at: https://scikit-learn.org/stable/modules/neural_networks_supervised.html, Accessed on:06/05/22

- 'Heaton Research' (2017) Available at: https://www.heatonresearch.com/2017/06/01/hidden-layers.html#:~:text=The%20number%20of%20hidden%20neurons,size%20of%20the%20input%20layer. , Accessed on:09/05/22

- 'sklearn.decomposition' Available at: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html, Accessed on:02/05/22

- 'sklearn.KNNImputer' Available at: https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html, Accessed on:03/05/22