MACHINE LEARNING COURSEWORK

TECHNICAL REPORT

Elixane Rey
Durham University
`elixane.c.rey@durham.ac.uk`

1 Data Exploration

1.1 Know

In this section I will explore the BraVL dataset. The table below provides summary statistics for the data:

Data	Samples	Features	Mean	SD	Min	Max
Brain (seen)	16540	561	-0.0673	0.9685	-9.1154	6.9173
Brain (unseen)	16000	561	-0.0759	1.5006	-30.9752	45.7661
Text (seen)	16540	512	0.0176	0.7019	-6.8098	14.9182
Text (unseen)	16000	512	0.0150	0.6884	-5.1338	11.0393
Image (seen)	16540	100	1.4575e-06	5.3636	-278.4464	619.5052
Image (unseen)	16000	100	-0.0343	4.2173	-33.6357	41.0052

The dataset was checked for completeness and consistency; there were no missing or invalid values in any of the classes.

1.2 See

Figures 1, 2 and 3 show the brain, text and image class distributions through histograms, as well as the corresponding outliers in the data. All the classes seem to have fairly normal distributions, with the brain data following the closest normal curve. Text data appears slightly skewed, as a few extreme outliers lie to the right-hand side of the bell curve.

Figure 4 shows the correlation between each of the classes in a single heatmap, featuring the correlation coefficients.

Figures 5 and 6 show a bar chart of class distributions across seen and unseen data, and can help detect class imbalances.

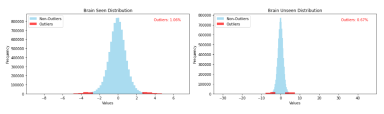
Figure 1: Brain Distribution

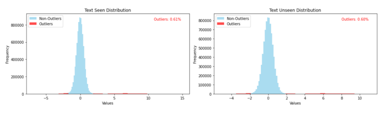
Figure 2: Text Distribution

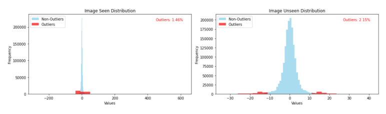
Figure 3: Image Distribution

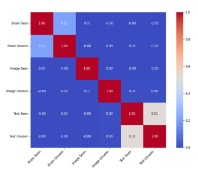
Figure 4: Class Correlation Heatmap

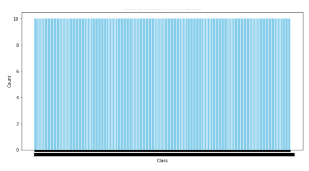
Figure 5: Seen Data Class Distribution

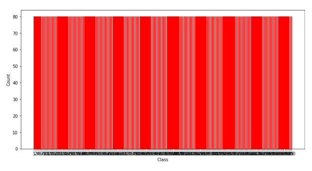
Figure 6: Unseen Data Class Distribution

1.3 Find

The K-Nearest Neighbours (KNN) model was chosen for this project, based on results from the data exploration, its simplicity and interpretability.

Outlier detection highlighted the presence of extreme values, particularly in the image class. However, these outliers are limited in number (<2.5% across all modalities) and manageable with a distance-based model like KNN, which is robust to such data points due to its reliance on proximity rather than distributional assumptions. The effect of outliers can also be minimised by tuning k (the number of neighbours considered).

The correlation analysis further demonstrated low to moderate inter-feature correlations, emphasizing that no single feature dominates the data representation. This independence among features suggests that distance-based methods, like KNN, which weigh all features equally in their decision-making process, are well-suited for the dataset.

Figures 5 and 6 suggest that there is a high class imbalance between seen and unseen classes. The bar charts show that seen data contains 1654 classes with 10 instances, while unseen data contains 200 classes with 80 instances. This could affect my KNN model performance as it might be biased toward the majority class, leading to higher misclassification in seen data. This issue will be dealt with in Section 2 of the report.

2 Model Implementation

2.1 Implement

As specified in the previous section, the selected sklearn baseline model is KNN. The dataset was divided into training and testing sets. The first 80% of samples in each class were allocated to the training set, while the remaining 20% were used for testing. Features from the brain, image, and text modalities were concatenated to create a combined feature vector for each sample. Outliers were identified and visualised (see Section 1) but were not removed, as their small proportion was not expected to significantly affect KNN performance. When implementing my initial model, I used a subset of 20 classes. This allowed me to create my code and deal with errors more efficiently.

In my custom model, predictions are based on the Euclidean distances between a test point and training points. This choice was made due to its simplicity and effectiveness in continuous feature spaces. For each test sample, the distances to all training samples were calculated. The k nearest neighbours are identified based on these distances. The class label with the highest frequency among the k neighbours is assigned to the test sample.

The primary hyper-parameter for KNN is the number of neighbours (k). Smaller values of k could make the model more sensitive to noise, but larger values of k (e.g., k = 15) might over-smooth the class boundaries, leading to a higher misclassification. To choose an optimal value for k, different values were manually tested (integers 1 to 20) until finding one which produced the highest accuracy. Based on the validation results, k=5 was chosen. This value balances the trade-off between model bias and variance.

2.2 Compare

The custom model performance was compared against the baseline model using the training behaviour and final test performance. Both models were tested on the entire dataset versus a subset of 20 classes, with the results displayed in the table below:

Model	Accuracy	Running Time
Baseline (whole dataset)	55.22%	0.0006 s
Baseline (20 classes)	68.33%	5.3 s
Custom (whole dataset)	56.21%	38 mins
Custom (20 classes)	70.62%	0.12 s

The baseline and custom model initially have very similar accuracy, both across the entire dataset and a smaller subset. However, the custom model is very inefficient compared to the baseline, especially when using the whole dataset. This is likely due to the distance calculations in the custom model, since the Euclidean distance method uses brute-force search, which can become very computationally expensive as the size of input data increases.

2.3 Improve

A major factor in improving the accuracy of the model is preprocessing the data. One of the ways in which the data was preprocessed was by normalisation, using a minmax scaler; all feature values were normalised to the range [0, 1] to ensure that differences in scale across modalities (brain, image, text) did not distort Euclidean distance calculations. The data was also normalised to zero mean and unit variance with a standard scaler, ensuring that features contribute equally to the distance calculation. Finally, PCA (Principal Component Analysis) was used on the data for dimensionality reduction, retaining 90% of the variance to decrease noise and computational complexity. PCA speeds up computation as there are fewer dimensions to calculate distances for, and improves accuracy as it removes irrelevant features that could introduce noise.

Within the model, a number of changes were also made to improve accuracy and running time. To address the class imbalance problem mentioned in Section 1.3, I implemented class weighting, which assigns higher weights to the minority class and reduces bias. To improve my model further, I added a more dynamic selection of hyper-parameter k. Instead of relying on a single k, the model evaluates performance across a range of k from 1 to 20 (using cross validation), and outputs the k with the highest accuracy.

Although all these features made slight improvements to the custom model, the preprocessing of data made the most significant change. When simply adding preprocessing (minmax scaler, standard scaler, PCA) to my initial model, accuracy shot up by 20%, and my code ran in 15 minutes for the entire dataset (compared to the original 38 minutes). However, class weighting did have a significant impact on model speed, cutting the total running time down to 2 minutes.

3 Result

3.1 Performance

The definitions of accuracy, precision, recall and F1-score are as below (TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \qquad F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Here is a table comparing accuracy, precision, recall and F1-score for the improved custom model and the baseline model:

Model	Accuracy	Precision	Recall	F1-score
Baseline (whole dataset)	55.22%	48.74%	55.22%	50.32%
Baseline (20 classes)	68.33%	72.23%	68.33%	65.89%
Improved Custom (whole dataset)	80.84%	83.75%	80.84%	80.69%
Improved Custom (20 classes)	99.17%	99.24%	99.17%	99.16%

The results show a high accuracy increase (around 25%) in my improved model compared to my original model, across the smaller subset of classes and the entire dataset. The precision, recall and F1-score are all consistent with accuracy, and show a balance between penalising false negatives and false positives. My custom model results show a higher precision rate than the baseline model, indicating a lower rate of false positives.

3.2 Visualisation

To illustrate results for my improved model, I created multiple confusion matrices on 20 random classes from the dataset, generating a different group of classes each time. These can be observed below in Figures 7, 8 and 9. A confusion matrix is a table that shows the model's performance by comparing its predictions to the actual results.

The results from the confusion matrices show that the model performs well overall, with most predictions aligning with the true classes. Across all three matrices, misclassification is minimal.

3.3 Ablation

The proposed improvements effectively address the problems of class imbalance, as reflected in the greatly improved accuracy and reduced misclassification in the confusion matrices. Inverse-distance weighting was successfully applied

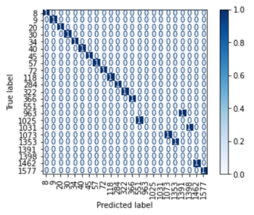

Figure 7: Confusion Matrix 1

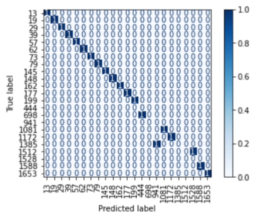

Figure 8: Confusion Matrix 2

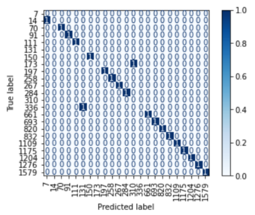

Figure 9: Confusion Matrix 3

by assigning a higher importance to closer neighbours, to stabilise the class imbalance. This made the model more robust in regions where class boundaries overlapped, as closer points are more likely to belong to the correct class.

Hyper-parameter tuning and preprocessing improved the model's ability to generalise unseen data. The cross-validation approach ensured that the chosen k is optimal across different subsets of the data. The use of PCA enhanced class separability by projecting data into a lower-dimensional space that retains relevant variance while discarding noise.

4 Paradigm

4.1 Paradigm

A traditional 80/20 train-test split ensures a balance between having enough training samples for learning and enough test samples for robust evaluation. However, experimenting with alternative splitting strategies can offer insights into the model's generalisation capabilities, especially for imbalanced datasets. Progressive splitting involves varying the proportion of training data (e.g., 50/50, 70/30, 90/10) to evaluate how the model's performance scales with the availability of training samples. This is useful for KNN to find the data sufficiency threshold, where adding data no longer improves accuracy. Progressive splitting allows for cost-benefit analysis in data collection, helping organisations decide the optimal dataset size to balance training costs and model performance.

4.2 Adjustment

To implement various train-test splits, a dynamic selection of k should be implemented for each dataset split during progressive splitting. The optimal k may vary based on the amount of training data, so cross-validation ensures consistency in performance as the experimental settings change. Gradually varying the proportions of training and test sets (e.g., 50/50, 60/40, 70/30, 90/10) allows for observations on how the model's performance scales with the amount of training data. Applying these splits on the entire dataset will create more precise results. This approach not only provides insights into the dataset's sufficiency for training but also helps identify potential thresholds where adding more data yields diminishing returns in terms of performance, enabling efficient resource allocation for data labelling.

4.3 Reflection

The table below presents results after running my custom model through different train-test splits:

Split	Accuracy	Precision	Recall	F1-score
50/50	75.15%	77.46%	75.13%	74.62%
60/40	76.21%	78.88%	76.21%	75.91%
70/30	76.89%	80.31%	76.89%	76.82%
80/20	80.84%	83.75%	80.84%	80.69%
90/10	77.47%	79.63%	77.47%	77.19%

These results show that the traditional 80/20 split is the most efficient for my custom KNN model.

References

Y. Long. Multimodal machine learning with brain, image, and text data, 2025. URL `https://colab.research.google.com/drive/1rejEQTW-J2bh88_DyVvHyboB-chSY0ZS#scrollTo=j-xrdgf7M0Pb`.

Long [2025]