1 A Complete Machine Learning Pipeline

In this report, I will outline the iterative machine learning pipeline procedure I have undergone, from data exploration to model implementation, results analysis and paradigm adjustment.

1.1 Data Exploration

i. Table of Summary Statistics:
Firstly, using Pandas data frames, I ensured that there was no missing data and that the dataset was complete.

For my initial data exploration, I utilised Pandas, NumPy and Matplotlib. I first investigated the data's summary statistics, giving the table below.

Table 1: Data Characteristics for Brain, Image, and Text Modalities

Modality	No. of Features	No. of Samples per Training Class	Range in Training Data Feature Standard Deviations	Range in Training Data Feature Means	No. of Test Samples
Brain	561	10	0.58 to 0.75	-0.29 to 0.08	80
Image	100	10	19 to 3.5	Approx. 0	80
Text	512	10	0.42 to 0.53	-0.70 to 0.85	80

The range and variance of image data is orders of magnitude greater than brain and text data. For models such as linear regression or KNN, which calculate Euclidean distance, features with larger ranges dominate, leading to a decline in accuracy. Therefore, image data must be scaled when combining modalities.

Using the mmbra library, I found that some features across brain, image and text had very large minimum and maximum values relative to their means and standard deviations, suggesting large outliers (e.g., brain feature 4: standard deviation 0.6, mean: 0.04, min: -3.57, max: 3.61). The upper and lower quartiles indicate relatively symmetric distributions for most features, although there is some skewness.

ii. Visualisations:

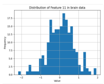
Figure 1.1

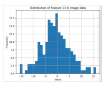
Figure 1.2

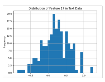
Figure 1.3

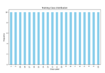
Figure 2.1

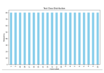
Figure 2.2

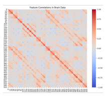
Figure 3.1

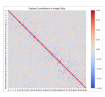
Figure 3.2

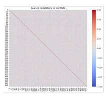
Figure 3.3

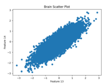
Figure 4.1

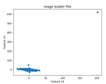
Figure 4.2

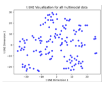
Figure 5.1

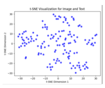
Figure 5.2

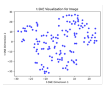
Figure 5.3

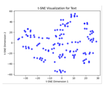
Figure 5.4

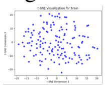
Figure 5.5

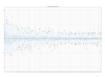
Figure 6.1

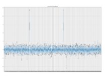
Figure 6.2

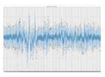
Figure 6.3

iii. Model Selection and Proposed Paradigm:
1. Feature Distributions: (Figures 1.1-1.3) Bar graphs of the first 20 features from brain, image, and text data show that features are approximately normally distributed, which is advantageous for models that assume Gaussian-distributed data (e.g., SVM, Linear/Logistic Regression).

2. Class Distributions:(Figures 2.1-2.2) Class distributions for training data (brain, image, text) are even, with 10 samples per class label. Test data has 80 samples per class label. However, the imbalance between training (10 samples per class) and testing (80 samples per class) could affect model performance due to insufficient data for meaningful learning in training.

3. Linear Relationships:(Figures 3.1–3.3) Heatmaps show strong positive correlations between numerically close features in brain data. These linear relationships suggest brain data could benefit from dimensionality reduction. Image and text data exhibit fewer linear correlations but demonstrate evidence of clustering, which could benefit clustering algorithms.

4. Non-linear Relationships: (Figures 4.1-4.2) Scatter plots of the first 20 features reveal potential clustering in image and text data. Clustering could improve performance using algorithms like KNN or Random Forest. However, scatter plots also reveal large outliers in image data, which must be addressed through clipping or removal. Brain data scatter plots confirm the presence of linear relationships between numerically close features.

5. Dimensionality Reduction: (Figures 5.1-5.5) Using t-SNE on training data, image and text data showed clear 2D clusters, but brain data showed no meaningful clusters. Multimodal combinations of data also performed well with t-SNE, suggesting clustering algorithms on multimodal (image and text) data could work well.

6. Outliers: (Figures 6.1-6.3) Bar charts of features (brain, image, text) highlight the number and magnitude of outliers. Image data features exhibit the most extreme outliers, reinforcing the need for clipping or scaling.

The proposed paradigm is zero-shot learning. Cross-validation will be used for my model to fine-tune hyperparameters and assess generalization performance. Models like KNN and Random Forest are well-suited to the image and text data due to the clustering tendencies observed. These algorithms handle non-linear relationships effectively.

1.2 Model Implementation

i. Implement - The model I chose to implement from scratch using NumPy arrays was Random Forest. This was based off my observations in 2.1. I believe that Random Forest Classification is the best suited algorithm for brain, image and text data on the whole, for five key reasons:

1) Handles High-Dimensional Data Well - For brain data, the Random Forest feature selection process (via tree splitting and feature importance ranking) is effective in reducing irrelevant information arising from redundancy between linearly correlated features such as the ones discovered in the brain data. For image and text data, random forest is well-suited to handling high-dimensional data without requiring significant dimensionality reduction beforehand.

2) Robust to Outliers - Outliers can significantly effect models like linear regression or SVM. On the other hand, Random Forest is robust to outliers because it relies on median splits and majority voting, which reduces sensitivity to extreme values, and each decision tree uses a bootstrapped random subset of features, minimising the influence of outliers on the final aggregated prediction across all trees.

3) Handles Non-linear Relationships Well - Random Forest is effective at capturing complex non-linear relationships (such as the clusters discoverd in the image and text data) due to its tree-based structure.

4) Insensitive to Feature Scaling - Random Forest doesn't rely on feature scaling unlike KNN, meaning large variances won't affect its performance negatively.

5) Works Well with Small Training Sets - I found that there were only 10 training samples per class. Random Forest can perform well on small training sets because it performs bootstrapping and averages predictions across different trees which reduces the risk of overfitting.

Implementation details: Each tree is trained on a bootstrap sample of the training data. A subset of features is selected at each split to introduce randomness and decorrelate trees. Each tree outputs a predicted class label and the final prediction is made by majority voting across all trees.

Data and Hyperparameters: - Starting by training the model on the first 20 classes of the brain, image and text modalities separately, I initially selected 50 trees with a maximum tree depth of 10, and a random selection of all 561 features to be considered at each split.

Validation: - I split the original training dataset into 80% training and 20% validation using train_test_split. The validation set is held out for final evaluation. Additionally, I performed 5-fold cross-validation on the training portion (80%) to evaluate the model's performance on unseen folds, ensuring that all training samples are utilized while reducing the risk of overfitting.

ii. Comparison Between Scikit-learn and Scratch Implementation of Random Forest
For my initial scratch implementation and sklearn model, I chose 50 trees and a maximum tree depth of 10:

Brain Data Performance

Metric	**Sklearn RF**	**Scratch RF**
Avg. Cross-Validation Accuracy	0.1357	0.1411
Validation Accuracy	-	0.0714
Test Accuracy	0.1333	0.1667
Training Time	0.25 s	23.77 s

Table 2: Comparison of Brain Data Performance

The scratch implementation required significantly more training time compared with the sklearn model due to inefficiencies, but it marginally outperformed in terms of test data accuracy. However, its cross-validation accuracy was lower than sklearn's, meaning that there is potentially some overfitting of the data.

Image Data Performance

Sklearn achieved higher test and cross-validation accuracy results than the scratch implementation, with the scratch implementation again being far slower. Both models struggled with lower recall and f1-scores for certain classes, suggesting potential issues with suboptimal hyperparameters.

Text Data Performance

Metric	Sklearn RF	Scratch RF
Avg. Cross-Val. Acc.	0.5429	0.4091
Val. Accuracy	-	0.4643
Test Acc.	0.5667	0.5167
Train Time	0.46 s	24.06 s

Figure 1: Comparison of Image Data Performance

Metric	Sklearn RF	Scratch RF
Avg. Cross-Val. Acc.	0.8786	0.7585
Val. Accuracy	-	0.8571
Test Acc.	0.8833	0.8667
Train Time	0.32 s	19.87 s

Figure 2: Comparison of Text Data Performance

The sklearn model consistently outperformed the scratch model, but both performed well on the text data. This makes sense since text data appeared heavily clustered after performing 2-d t-SNE. Again, certain classes had 0% recall and f1-score, indicating suboptimal hyperparameters. On the whole, the scratch implementation is significantly slower compared to sklearn. The sklearn model also handles class imbalances better, reflected in higher f1-scores and recall for minority classes.

iii. Improvements to Scratch Implementation - PCA dimensionality reduction: Using PCA dimensionality reduction, I found that 77 features, 44 features and 64 features of the brain, image and text data respectively accounted for 95% of their feature variances. By selecting the highest variance features and discarding the rest, we will increase the convergence speed (since the model is trained on fewer features) while encapsulating most of the data's information. **Feature scaling:** In 2.1, I found that each of the data modalities had vastly different ranges and standard deviations, and that the characteristic statistics of individual features within each modality also ranged greatly. Although Random Forest is not directly affected by varying feature scales, feature scaling enables PCA to perform better. I therefore will implement feature scaling before PCA in my pipeline.

Hyperparameter tuning: I could perform grid search to improve the selection of my hyper-parameters, but when my model takes 10 minutes to complete 5-fold cross-validation, produce the confusion matrix and convergence speed etc., grid search with even a small number of options per parameter (e.g. 4-5) produces a vast number of combinations of hyper-parameters and results in hundreds of hours in training time. I will therefore tune the parameters by hand. Hyper-parameter tuning improves generalization and model stability. I will increase the number of trees from 50 to 100 and the maximum tree depth from 20 to 10 to combat over-fitting.

1.3 Result Analysis and Visualisation

Metric	Sklearn RF (Start)	Scratch RF (Start)	Sklearn RF (End)	Scratch RF (End)
Test Accuracy	0.1333	0.1667	0.25	0.20
Precision (Avg)	0.12	0.11	0.22	0.12
Recall (Avg)	0.14	0.17	0.25	0.20
F1-Score (Avg)	0.13	0.14	0.23	0.15
Cross-val. Accuracy	0.1357	0.1411	0.19	0.09
Training Time	0.25 s	23.77 s	0.54 s	46.63 s

Table 3: Brain Data Model Improvements

Metric	Sklearn RF (Start)	Scratch RF (Start)	Sklearn RF (End)	Scratch RF (End)
Test Accuracy	0.5667	0.5167	0.75	0.67
Precision (Avg)	0.56	0.48	0.84	0.67
Recall (Avg)	0.57	0.52	0.75	0.67
F1-Score (Avg)	0.56	0.49	0.74	0.64
Cross-val. Accuracy	0.5429	0.4091	0.59	0.45
Training Time	0.46 s	24.06 s	0.44 s	44.52 s

Table 4: Image Data Model Improvements

i. Performance Before the table analysis, I will lay out some definitions: Accuracy: Measures the overall correctness of the model. Precision: Proportion of true positive predictions out of all positive predictions. Recall: True Positives / All Positives. F1-Score: Balances precision and recall, providing a harmonic mean of the two metrics.

On the whole, sklearn still performs better after making improvements to the scratch model. Particularly, the scratch model is worse at cross-validation suggesting underfitting or overfitting. Scikit-learn model has slightly better generalisation. However, when trained on the text data, the two models perform very similarly (apart from their training times).

ii. Visualisations - I can't get it to go in the right place. The relevant confusion matrices are below the tables.

iii. Ablation - The training data had only 10 samples per class, while the test data had 80 samples per class. This imbalance affected the model's ability to generalize well during training. Therefore, I used enhanced pre-processing techniques before training the model on the data. For example, by applying standard scaling and PCA, I aimed to

Metric	Sklearn RF (Start)	Scratch RF (Start)	Sklearn RF (End)	Scratch RF (End)
Test Accuracy	0.8833	0.8667	0.95	0.95
Precision (Avg)	0.87	0.83	0.96	0.96
Recall (Avg)	0.88	0.86	0.95	0.95
F1-Score (Avg)	0.87	0.84	0.95	0.94
Cross-val. Accuracy	0.8786	0.7585	-	0.79
Training Time	0.32 s	19.87 s	0.84 s	38.56 s

Table 5: Text Data Model Improvements

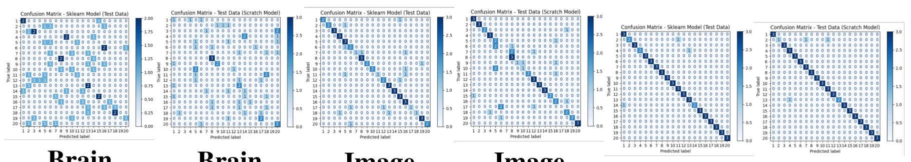

Figure 3: Confusion matrices for Brain, Image, and Text datasets using Sklearn and Scratch models.

improve the model's ability to learn meaningful patterns even with limited data. Furthermore, I used cross-validation to help improve the model's generalization ability, even with small training data. Despite these improvements, the performance of the Scratch RF model still lags behind the Sklearn RF model, particularly in the Brain data (test accuracy of 0.25 vs 0.20). However, the Sklearn RF model shows an increase in accuracy (from 0.1333 to 0.25) as a result of these improvements, demonstrating better handling of the imbalance problem.

Dimensionality and scaling issues were mitigated with standard scaler and PCA, improving model accuracy and stability. The clustering relationships discovered in section 2.1 were effectively handled by the Random Forest Algorithm. Long training times were initially reduced by PCA but once I increased the number of trees to improve accuracy, training times shot up again.

1.4 Paradigm Adjustment

i. Paradigm - For my paradigm adjustment, I switched from Zero-Shot Learning with Random Forest to Few-Shot Learning with the sklearn KNN clustering algorithm. In Few-Shot, the model is trained on a very small number of samples from the labelled class and made to generalise well to other samples in the class. The training set is called the "Support" set and the testing set is the "Query" set. The train_test_split occurs at the class level to ensure that all samples are in "Query" set or "Support" set, but never both. The real-world value of zero-shot learning is that it's expensive and time consuming to label lots of data, especially in industries such as medical and drug research.

ii. Adjustment - Instead of splitting the data into seen and unseen labels, I can take 2-3 samples from each of the previously seen labels as the Support set, placing the rest of the samples from each seen label in the Query set. I'll train on the Support set, potentially using techniques such as data augmentation to combat the shortage of data.

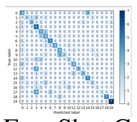

Figure 4: KNN Few-Shot Confusion Matrix

Metric	**Value**
Accuracy	0.51
Macro Average Precision	0.61
Macro Average Recall	0.51
Macro Average F1-Score	0.50
Weighted Average Precision	0.61
Weighted Average Recall	0.51
Weighted Average F1-Score	0.50

Table 6: Summary metrics for KNN on few-shot learning dataset.

iii. Reflection - The recall is lower than precision, indicating that the model is better at avoiding false positives than identifying all true positives. This could be because KNN is sensitive to the choice of neighbors and the feature space. It may correctly classify close neighbors (high precision) but fail to cover the broader space (low recall).