Cross Paradigm Analysis of Things-EEG Dataset

Rose Laird, Durham University

January 23, 2025

1 Dataset Exploration

The dataset I used was produced from a large study, where participants would be shown an image of a certain 'Thing' and electroencephalogram signals were measured. I aim to classify these 'Things' using the data provided.

1.1 Statistical Findings

To best analyze the dataset I first did some statistical exploration to inform my decision. The part of the larger Things-EEG dataset that I am analyzing contains 140 training samples and 60 testing samples, evenly split across 20 classes, with each class represented by 7 training samples and 3 testing samples. Each class represents a 'Thing' or label. They comprise of three different aforementioned modalities:

- **Brain features**: 561 EEG-based features.
- **Image features**: 100 features from the image shown to the participant.
- **Text features**: 512 features of the English description of the 'Thing'

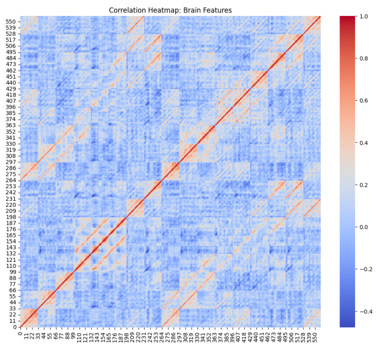
Stacking these results into 1,733 dimensions per sample presents challenges such as potential redundancy and computational overhead. The statistics below summarized in Table 1show reasonable properties suited for machine learning, but brain features exhibited lower variance compared to image features, suggesting potential redundancy. These findings indicate that feature scaling and dimensionality reduction are necessary to ensure efficient training and better generalization.

Dataset	Mean of Means	Std of Means	Mean of Std Devs	Std of Std Devs	Mean of Ranges	Std of Ranges
train_brain	-0.046718	0.553780	0.759917	0.127805	4.067105	0.774885
train_text	0.012790	0.522642	0.465467	0.114634	2.338500	0.594113
train_image	-0.097658	0.736822	3.959818	2.270744	22.317346	11.352850

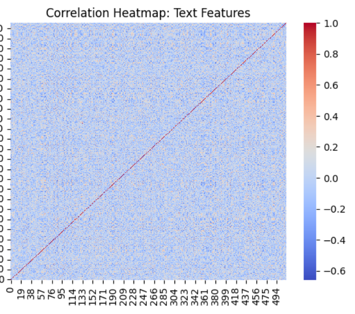
Table 1: Aggregate Statistics of Different Datasets

1.2 Visualizations

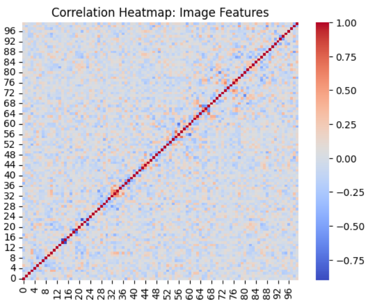
Feature correlation heat maps reveal minimal strong correlations, highlighting the independence of most features 1. PCA analysis indicated that 95% of the dataset variance could be explained with approximately 70 principal components, emphasizing the high dimensionality redundancy 2.



(a) Brain Features



(b) Text Features



(c) Image Features

Figure 1: Visualizations of Different Features

1.3 Findings and Implications

The dataset's balanced class distribution supports consistent model performance evaluation. However, its high-dimensional, multi-modal nature introduces risks of over fitting and computational inefficiency, particularly when

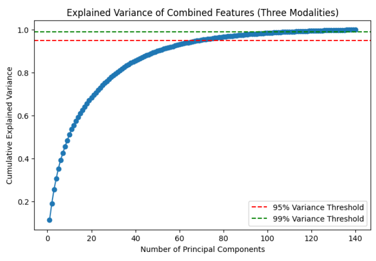


Figure 2: Principal Component Analysis (PCA) Visualization

combining modalities. PCA is a necessary step to address these issues by reducing redundancy and retaining only the most significant features, improving both runtime and model performance.

2 Random Forest Implementation

2.1 Implementation

Random Forest is an ensemble learning algorithm that constructs multiple decision trees during training and aggregates their predictions to produce a final result. Each tree in the forest is trained on a different bootstrap sample of the dataset, and randomness is introduced by selecting a random subset of features for each split in the trees. This dual-randomization mechanism enhances the model's robustness and reduces the risk of over fitting, which is a common issue in single decision trees. The algorithm is particularly well-suited for datasets with high-dimensional features and mixed modalities, and so it is my choice for the Things-EEG dataset.

Other models were less immediately appropriate for this task. Support Vector Machines (SVMs) and Logistic Regression tend to be more suited to binary classification problems and SVMSs can struggle with scalability in high-dimensional, multi-class datasets like the one I am using. K-Nearest Neighbors (KNN), while more suited for multi-class and multi-modal data can quickly become computationally expensive in high-dimensional datasets. Random Forest, by contrast, can handle these challenges efficiently and is better equipped for multi-class classification while providing robust performance in high-dimensional feature spaces.

2.1.1 Hyperparameter Tuning

To manage the high dimensionality of the dataset, as seen in Data Exploration, and to reduce model runtime, I applied Principal Component Analysis (PCA) before training the Random Forest. Then to optimize the hyperparameters of the Random Forest model, I used a grid search method. The following hyperparameters were tuned:

- **Number of Trees (`n_estimators`):** 10, 20 and 50 to balance model performance and computational efficiency.
- **Tree Depth (`max_depth`):** No depth (None), 10 and 20 to control complexity and prevent over fitting.
- **Minimum Samples Split (`min_samples_split`):** 2, 5, and 10 to manage the granularity of splits and control overfitting.

Using 5-fold cross-validation, the grid search optimized the hyperparameters to 150 trees, a maximum depth of 35, and a minimum split size of 2, balancing accuracy and computational efficiency despite the high dimensionality.

2.2 Initial Comparison

The final accuracy of the initial custom model was 0.65, lower than the scikit-learn model which achieved an accuracy of 0.8, The custom model also required a significantly longer runtime, taking 75 seconds compared to the scikit-learn model's 15 seconds. Additionally the scikit-learn model reached an accuracy of 0.40 at 20 trees, whereas the custom model achieved a slightly higher accuracy of 0.63 at the same point, highlighting the differences in performance and efficiency between the two implementations.

2.3 Improvements

To increase accuracy and also reduce training time I employed the following improvements:

- Tree Splitting Optimization: To better find the best split in a Decision Tree, I tried evaluating only the midpoints between unique values of a feature rather than every possible threshold. This reduces the number of calculations needed during training, speeding up the model without losing accuracy, increasing efficiency.
- Entropy calculations:I replaced Python loops with numpy's efficient vectorized operations. Probabilities are computed directly from a histogram of class labels and avoids numerical errors by adding a small constant to the logarithm function, improving training speed.
- Early Stopping Criteria: The tree stops splitting further if the potential improvement in information gain is minimal, or if the size of the data at a node is too small to make reliable splits. This reduces unnecessary computation, shortens training time, and improves the model's generalization to unseen data, preventing overfitting.
- Parallelization: To reduce training time I implemented Python's joblib library, which trains multiple trees at the same time on separate CPU cores, as each tree is independent of the others.
- Weighted Voting: To increase accuracy I incorporated weighted voting, such that each tree's contribution is proportional to its accuracy on out-of-bag (OOB) samples(samples it's not trained on). This ensures that better-performing trees have a greater influence on the final prediction.
- Isolation Forest Outlier Detection: I used a pre-built Isolation Forest algorithm to improve data quality and reduce noise, which works by identifying samples with abnormal feature patterns and removing them before training the model. This amounted to 5% of the training samples and resulted in a significant accuracy improvement from 71% to 87%. 2

Features Added	Accuracy	Runtime
None (PCA)	0.650	85s
Improved best split	0.670	55s
Entropy	0.67	41s
Early Stopping	0.75	38s
Parallelization	0.75	20s
Weighted Voting	0.75	30s
Isolation Forest	0.87	40s

Table 2: Cumulative Improvements, Accuracy, and Runtime of the Model

3 Model Comparison and Evaluation

3.1 Performance Metrics

The performance metrics for the three models are summarized in Table 3. Logistic Regression demonstrates the highest performance across all metrics, achieving an accuracy of 0.97, while the Scikit-learn Random Forest marginally outperforms the custom Random Forest in all metrics. This highlights the efficiency and optimization of the sci-kit learn library for general use cases. The high f1 scores indicate an effective balance of precision and recall.

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)
Custom Improved Random Forest	0.83	0.82	0.83	0.81
Scikit-learn Random Forest	0.85	0.83	0.85	0.82
Logistic Regression	0.97	0.97	0.97	0.96

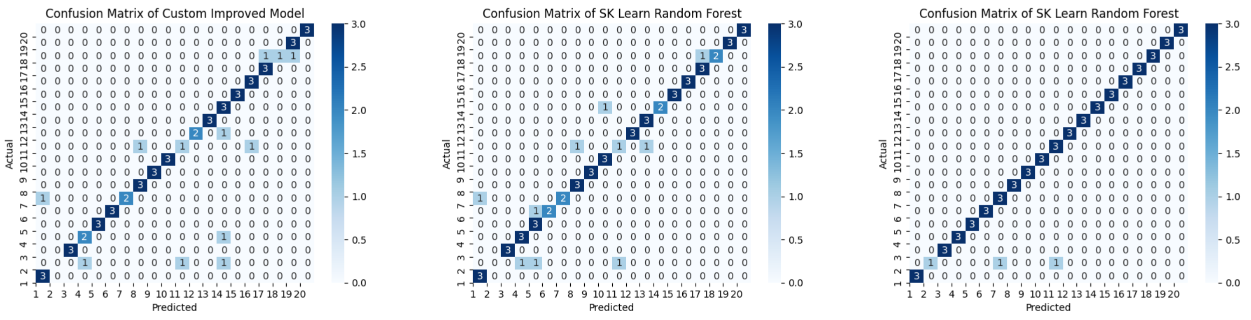
Table 3: Comparison of performance metrics for the evaluated models.

3.2 Confusion Matrix Visualization

The confusion matrices for each model are visualized in Figure 3. Logistic Regression exhibits the highest diagonal dominance, indicative of superior classification accuracy. All 3 models struggled to label the second class, which suggests inherent data difficulties with that class, perhaps suggesting overlapping features.

3.3 Ablation

These results demonstrate partial success in tackling the potential problems of the dataset introduced earlier. This dimensionality posed risks of redundancy and over fitting, particularly in the brain and text features. By reducing the dimensionality while preserving the most important variance in the data, PCA ensured that the Random Forest model focused on the most meaningful patterns, improving both efficiency and performance. My outlier reduction technique



(a) Custom Random Forest (b) Scikit-learn Random Forest (c) Logistic Regression

Figure 3: Confusion matrices for the evaluated models.

further improved accuracy, as shown in my cumulative iteration log. However it still fell short of both Sci-kit learn models, suggesting it would benefit from further improvements such as advanced feature engineering or an ensemble approach.

4 Active Learning

4.1 Paradigm

To facilitate an Active Learning paradigm, I re-split the training dataset in a stratified way such that the training set had 20 samples, one from each class, ensuring proportional representation. The remaining 120 samples were allocated to the unlabeled pool for iterative querying. Entropy-based uncertainty sampling was used to select the most informative samples for labeling, reflecting real-world scenarios where data labeling may be resource or cost-intensive. This may be especially important for the context of this dataset, as it could allow for an increased training set incorporating other non-labeled EEG datasets that might be available in the neuroscience field, overcoming the resource-intensive process of more human trials.

4.2 Adjustment

To accommodate active learning, I had to rewrite and edit my improved Random Forest Model. The following adjustments were made:

- **Add Aggregate Votes Function** I introduced a `predict_proba` method to compute class probabilities by aggregating tree-level predictions. This enabled the calculation of entropy for uncertainty-based querying.
- **Batch-based Processing:** 10 samples were queried per iteration and 10 iterations took place to increase performance over training.
- **Dynamic Size Calculation:** I calculated initial size of the sample dynamically to ensure it included at least one sample per class - no zero-shot learning!

4.3 Reflection

The model was trained using an active learning framework, showing consistent improvements in test accuracy across iterations, with a final accuracy of 90%. This underscores the efficiency of entropy-based uncertainty sampling in boosting generalization while minimizing labeled data requirements.

Initially, the custom model achieved an accuracy of 0.22. By implementing entropy-based uncertainty sampling and iteratively querying uncertain samples, accuracy improved to 0.65. Stratified splitting helped mitigate class imbalance, ensuring robust learning from the initial dataset. Iterative querying enhanced learning efficiency by focusing on challenging samples, demonstrating the real-world value of active learning for limited labeling budgets.

Iteration	0	1	2	3	4	5	6	7	8	9-10
Accuracy	0.22	0.17	0.23	0.25	0.32	0.35	0.45	0.60	0.53	0.65

Table 4: Test Accuracy Over Active Learning Iterations