BraVL Analysis

Hari Patel
github.com/Hari-P-22121
www.linkedin.com/in/hari-patel-37618a181

1 Introduction

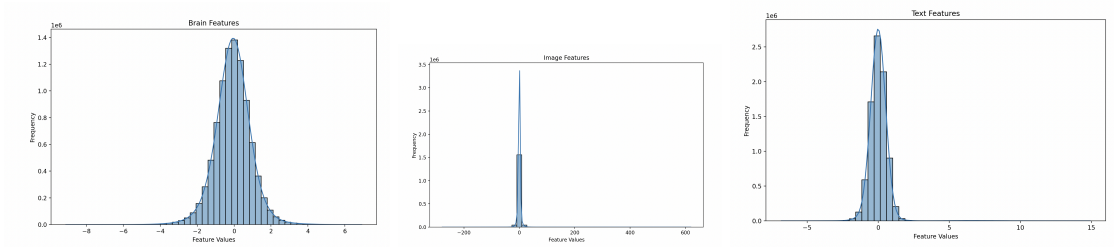
The goal of this project is to analyze the BraVL (https://github.com/ChangdeDu/BraVL) dataset. This trimodal dataset that analyzes visual data (images), textual data (image descriptions), and brain signals as EEG scans (stimuli response to the visual data). We must attempt to classify these embeddings to a label. Our label is a word that is encoded into an index of numbers. When given all three modalities, we should be able to find a pattern in the brain response, visual information, and textual description to classify it to an object in our labels.

2 Data Exploration

We will use Pandas to create dataframes, and from there we will use Matplot to explore the distribution of the data. We will look at the three features, image, brain, and text.

2.1 Distribution of Features

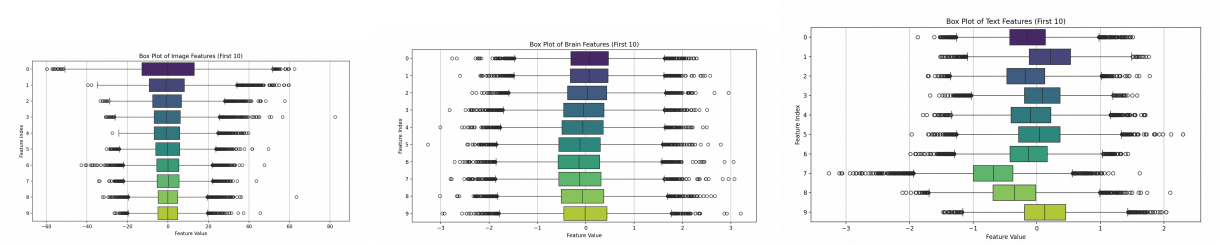
SEE



We see a different distribution that causes an imbalance within the data especially for the image data.

2.2 Outliers in Features

SEE



Using a box plot for visualization, we see outliers in the data we must address.

2.2 Data Preparation

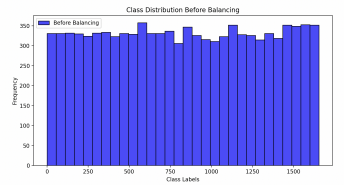
After taking our seen data, we must preprocess this before we split into training and testing. We will perform standardized scaling on the data. To fix our outliers, we will use an isolation forest which isolates anomalies using decision trees to identify outliers, after this we can use a mask with indices to remove the

discovered outliers. We will also use PCA reduction to make the data more efficient using `n_components=0.95` to retain 95% variance.

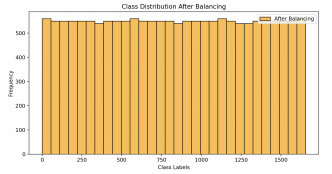
2.3 Data splitting

We can finally combine our three datasets and perform a stratified split. This ensures that the class proportions are reserved which helps with imbalanced data like ours. We will perform an **80/20** split on the data. We also want to fix class imbalance by using Random Over Sampling, an oversampling technique. Notice the change in distributions.

SEE Before RandomOverSampling



SEE After RandomOverSampling

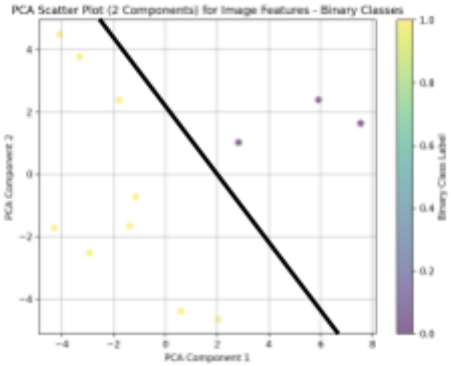


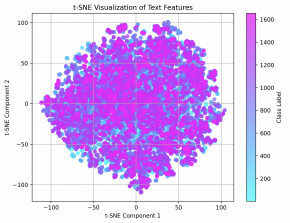
2.5 Model choosing and preparation

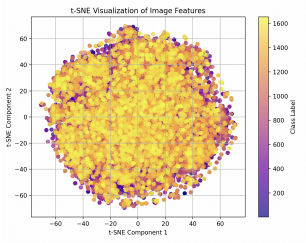
KNOW

When we use PCA to further reduce our dataset to map our data onto a 2D space. When we do this, we see linear separability, suggesting Logistic Regression is a good choice. We can see below the PCA reduction for Image Features using `n_components = 2` that the two components are linearly separable when mapped onto a 2D space.

SEE







FIND

A line is drawn to show the linear separability. Our TNSE plots show sub datasets to have clusters apart from others.

3 Baseline and Custom Model Implementation

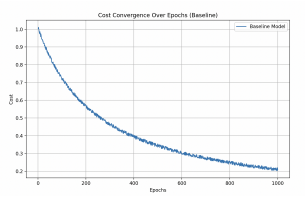
3.1 Logistic Regression Classification Model

Our goal will be to take a feature and classify it to a single label. Despite data variables not being continuous enough for regression models for relationships, we can use logistic regression for classification by using a function to output probability scores using softmax after the regression. This is similar to how Transformer Models work by predicting a probability distribution to find out which token is most likely to come next. However, this project does not explore deep learning, so we can use a simpler

version like logistic regression with a single layer to attempt to classify a probability score to take all three modalities (brain, image, text), and classify it to a label.

3.2 Baseline Model

IMPLEMENT



We will implement the baseline model using scikit-learn and analyze our model initially with accuracy to see if the model was a good selection. Upon training the model, we achieved a **55%** accuracy everytime with the baseline. This is a good start, let's see how our custom model can improve this. Our accuracy is determined by comparing our one hot encoding labels. We also see that our convergence is decent but not very good along with our recall and support. We will also try to import the **Baseline F1 (Weighted): 0.54, Recall (Weighted): 0.55, Precision (Weighted): 0.53** scores.

3.3 Custom Model

IMPLEMENT

For our loss function, we use binary cross entropy loss which finds the error difference using logarithms. We will use this with our gradient descent function to make the adjustments.

Loss function (Helps us quantify how different the actual and predicted is):

$$J(\Theta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(\widehat{y}_i) + (1 - y_i) log(1 - \widehat{y}_i]$$

Gradient Descent function (Finds the error difference and modifies the model accordingly):

$$\frac{\partial J}{\partial \Theta_j} = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)x_{ij}$$

Gradient Descent adjustment equation with the loss function:

$$\theta_j = \theta_j - \eta(\frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)x_{ij})$$

We will also use a standard scaler to balance the data better.

COMPARE

The accuracy for our custom model is **34.28% (Precision: 0.32, Recall: 0.34, F1: 0.33)** which is lower than the baseline model. The problem is Logistic Regression is very sensitive to outliers and imbalance in data we must tackle.

3.6 Custom Model Improvement

IMPROVE

L1 and L2 regularization adds a penalty term in the loss function or in this case our cost_function and gradDes function. Regularization is good for our dataset since it has multiple features and possibly multicollinearity. This penalty term we add to the final loss value released by our cost function will help prevent overfitting. This also helps larger weights from having too much influence on the overall model. The key difference between **L1** and **L2**:

L1: Sums the absolute values of the weights in the cost function

L2: Sums the squared values of the weights in the cost function

L1 Regularization term: $\frac{\lambda}{m} sign(\theta_j)$

L2 Regularization term: $\frac{\lambda}{m}\theta_j$

Overall Gradient Descent Function: $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{ij} + \frac{\lambda}{m} sign(\theta_j) + \frac{\lambda}{m}\theta_j$

We will also attempt to initialize better weight values. We want to create a matrix size of the number of features by number of classes, and initialize a standard normal distribution so our weight values are valid decimals. We can then scale them by 0.01 to make them smaller and have a mean of 0 and standard deviation of 0.01. This is important to start training our model closer to the actual possible values giving it a headstart, decreasing too much reliance on the convergence and overall model. We also shuffle the indices every 10 epochs. At the same time, we calculate our loss function every 100 epochs and make the adjustments. We also can initialize better weights rather than starting from a random value.

3.7 Mini Batch Processing

By using smaller batches that we shuffle, our model becomes more efficient and also becomes better at generalizing data through variation.

3.8 Outlier Removal

We can use an Isolation Forest which can isolate anomalies, helpful for multimodal features like ours.

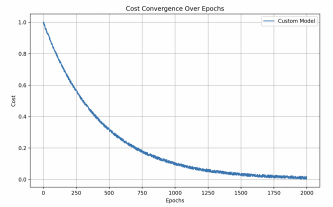
3.8 Paradigm Design and Data Splitting

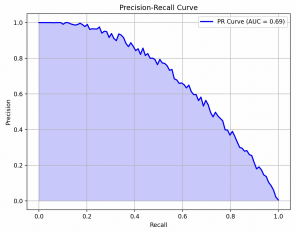
PARADIGM and ADJUSTMENT

We can try different stratified splits, such as **70/30** split instead. Our final custom model accuracy becomes **56.48%**, which is higher. Paradigm shifts like zero shot, few shot, federated learning could be considered but not required since we are performing logistic regression and we use other methods to fix imbalance before using a stratified split.

Final analysis of custom model results

PERFORMANCE AND REFLECTION





$$Precision = \frac{True\ Positive}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

We can't use few-shot learning. This is because when using one vs all, it is important to give each feature as much information as possible, decreasing the information given makes it difficult for the model to distinguish the characteristics, therefore we will not use either of these. After seeing our accuracy improve, we also want to look at more metrics such as Precision, Recall, and F1-Score. This can be calculated using scikit-learn. We will then find the weighted average across all the features classifications. We can see that the problem identified with outliers and imbalance data was not only tackled with isolation forest but with regularization, mini batch processing, and data splitting. RandomOverSampling helped us fix the imbalance as well. We can see not only a higher accuracy but better precision, recall, and F1 score.

Baseline Model	**Custom Model Final**
Accuracy: 55 %	Custom Accuracy: 56.48 %
Baseline F1 (Weighted): 0.54	Custom Precision (Weighted): 0.62
Baseline Recall (Weighted): 0.55	Custom F1 (Weighted): 0.55
Baseline Precision (Weighted): 0.53	Custom Recall (Weighted): 0.56