CS771 Mini Project-1 Fall-24

Group 29: The Outliers

Himalaya Kaushik
241110029

Keshav Banka
241110032

Pritindra Das
241110054

Tafazzul Nadeem
232110401

Yuvraj Raghuvanshi
241110084

1 Introduction

Our project is on binary classification task of Machine Learning where we are provided with three datasets with train, validation, and test splits, which only differ in terms of features being used to represent each input from the original raw dataset. There are two tasks to perform, first, to come up with a best model for each dataset which can generalize well for binary classification task requiring small amount (20%, 40%, 60%, 80%, 100%) of training data. The second task is to use the three datasets altogether to find a model for the classification task and analyze if accuracy can be enhanced by combining the features of the three datasets, with a constraint of 10k learnable parameters in both the task.

This report which is divided into four sections. In Section 2 we give a brief description of the three datasets with its dimensionality and datatypes. Section 3 describes our approach till evolution to the best model for each of the datasets in Task 1 and for the combined dataset in Task 2 with results in the form of tables and plots along with its analysis. We conclude the report in section 4 summarizing the findings of our project.

2 Dataset Description

2.1 Dataset-1 (Emoticons Dataset)

Each input in this dataset has 13 emoticons in its characteristics that correspond to the output label of either 0 or 1. We discovered 214 distinct emoticons after preprocessing the training dataset. In order to further evaluate the data, we looked at the frequency of each emoticon and whether they had any relative meaning.

2.2 Dataset-2 (Deep Features Dataset)

This is a transformation dataset where each emoticon of the 13 emoticon input in the previous dataset is transformed into a 786 dimensional input of float datatype using a deep neural network. Since the features are transformed to a very high dimensional space, dimensionality reduction techniques like pooling can be used to abide the constraint on learnable parameters.

2.3 Dataset-3 (Text Sequence Dataset)

The text sequence dataset consists of 50-digit input strings, where each string serves as a feature representation for binary classification (label $\in \{0, 1\}$). The dataset is formatted as a CSV with columns 'input_str' and 'label'.

3 Models, Experimentations, and Results

3.1 Emoticon Dataset

Vectorizer method- One method for converting emoticon data into numerical representations that can be utilized in this dataset is the TF-IDF(Term Frequency-Inverse Document Frequency) [3] vectorizer. The conclusions we reached after experimenting serially with various learning strategies and after validating the results are displayed in Table-1.

ML Models	Valid Acc on 100% of i/p	Inference
KNN	0.48	K values from 1-31, best accuracy for k=13
Decision Tree	0.49	best model inferred at depth = 25
Logistic Regression	0.45	4 solvers, liblinear solver yielded the best
Gaussian Naive Bayes	**0.51**	Used categorical NB but failed
SVM	0.49	4 kernels, linear kernel showed best among them

Table 1: Valid. accuracies after training inputs of the Emoticon Dataset (vectorized)

As observed, the accuracies of all the models after vectorizing the data are approximately 0.5. This performance is close to that of a random classifier, indicating that the models are not effectively distinguishing between the classes and are performing only marginally better than random guessing.

One hot encodings- Initially, we tested various methods like decision trees and K-nearest neighbor, with one hot encoding technique achieving better accuracy than before. Through experimentation, we found that Support Vector Machine (SVM) and Logistic Regression yielded the best results for identifying the categorical output. Before trying neural networks we got the best accuracy results from implementing logistic regression and SVM. Further tuning the hyperparameters of the models and using different penalty norms and solvers we get the following results as shown in Table-2:.

ML Models	Valid Acc on 100% of i/p	Inference
KNN	0.58	K values from 1-31, best accuracy for k=13
Deciosn Tree	0.77	best model inferred at depth = 31
Logistic Regression	0.89	Liblinear-L1 solver-penalty yielded best accuracy
Gaussian Naive Bayes	0.81	Without priors and 1e-09 variance smoothing
SVM	**0.90**	Linear kernel with 80% train data
XG Boost	0.63	Logistic objective and depth = 6
Random Forest	0.82	At max depth=19, estimators=252

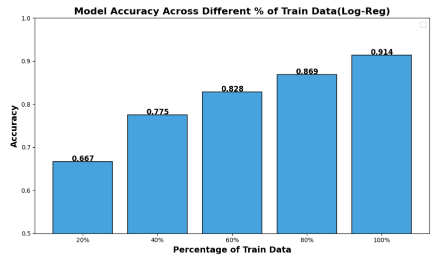
Table 2: Valid. accuracies after training inputs of the Emoticon Dataset (One-hot encoded)

According to Fig. 1(a), the Logistic Regression model works best when all of the training data is used, and accuracy rises as the percentage of training data increases. Furthermore, Fig. 1(b) suggests that the optimal liblinear solver for the emoticon dataset is one that uses the L1 norm as the model's penalty, yielding accurate results. Similar to this, Fig. 1(d) illustrates that when employing the SVM learning technique, the models provide good accuracy when we use 60–100 % of the training data. Additionally, based on the binary classification data, we may infer from Fig. 1(c) that the most accurate SVM kernel for this dataset is the linear kernel.

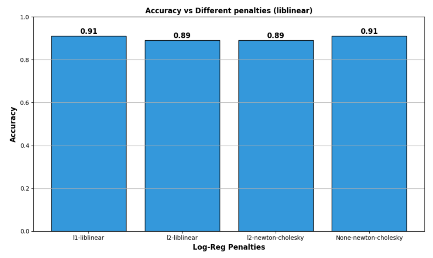
Colab Notebook link (Vectorizer and One hot encoding)- Click here

Best Model (Embedding model)- The neural network for binary classification consists of several components with specific functions. Initially, the input is represented as integer indices, resulting in a tensor with a shape of (7080, 13). The flattening layer then reshapes this data into dimensions of (7080, 1). A fully connected (FC) layer applies a ReLU activation function to transform the flattened input into an 8-dimensional space, followed by a dropout layer for regularization. The second FC layer maps the output to a single value, which is passed through a sigmoid activation function for classification. A forward method defines how the input data flows through the network: embedding $\rightarrow$ flattening $\rightarrow$ fully connected layers $\rightarrow$ sigmoid output.

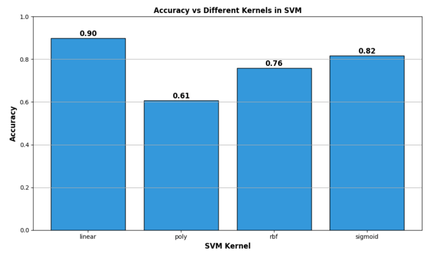
The model has a total of 6,617 trainable parameters. After training with varying proportions of the training data, the model demonstrated optimal performance when trained on 60-100% of the dataset



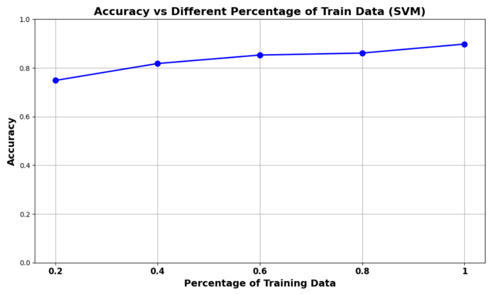
((a)) Accuracy vs Train Data (Log-Reg)



((b)) Accuracy vs Solvers (Log-Reg)



((c)) Comparison of Kernels (SVM)



((d)) Accuracy vs Train Data (SVM)

Figure 1: Results on Logistic Regression and SVM models on Emoticons Dataset

as shown in Table-3. Consequently, a simple neural network using integer embeddings was selected as the most suitable model for this dataset.

Model	**20% of i/p**	**40% of i/p**	**60% of i/p**	**80% of i/p**	**100% of i/p**
Embedding Model	0.955	0.948	0.955	0.953	**0.957**

Table 3: Validation Accuracy of best model across incremental training input sizes on Emoticon Dataset

Colab Notebook link (Embedding Model)- Click here

3.2 Deep Feature Dataset

The deep feature dataset consists of 9984 (13x768) dimensional input, hence the first thing we tried was to reduce its dimension with pooling techniques like mean or average pooling and max pooling popularized by Yann LeCun et al. [2]. Averaging and max along the columns of the 13x768 dimensional inputs reduced the input dimension to 768. After reduction, a simple logistic regression model with Binary Cross Entropy loss was trained on the training dataset and tested on the validation dataset. Even after trying different learning rates, and optimizers, the model was not able to learn, and validation set accuracy hovered around 50%. The results are reported in Table 4.

The linear logistic regression model not able to learn the distribution of the reduced dimension inputs led us to try injecting non linearity into our model. A model with hidden layer of 64 dimension with leaky relu activation between the 768 dimensional input and 1 dimensional output was trained on the pooled train dataset. The accuracy even further stooped to 45% hinting at pooled data not linearly seperable . We experimented with different learning rates, changing the hidden layer dimensions, having multiple hidden layers etc but to no avail. The accuracy did not improve further the 50% mark. Moreover, the hidden dimension approach was also making the model go beyond the 10k learnable parameter mark. The results are reported in Table 4.

At this stage it was clear that combining the embeddings is leading to complete loss of information and hence not a way forward. In the next approach we used a sophisticated dimensionality reduction technique, the Principal Component Analysis (PCA) by Pearson [1]. The 13x767 dimensional input vectors from the train set was flattened to form a long 9984 dimensional vector and then PCA analysis was done to capture the most significant 300 dimensions. After that, we used a simple Logistic Regression model learn the hyperplane. The results were astonishing as we were getting an accuracy of 93-94% after few epochs by doing mini batch gradient descent with Adam optimizer and cross entropy loss. The results are reported in Table 4.

Although we had reached respectable enough accuracy on the validation set we tried to further improve our model since the PCA analysis model was only having 301 learnable parameters, so there was room for improvement. Since, the flattened vector used for PCA was having 9984 dimension which was below the 10k parameter constraint, we went for a linear Logistic Regression model on the full flattened vector. The validation accuracy touched 98% in this setting with 9985 learnable parameters in total. We experimented with 20%, 40%, 60%, 80% and 100% of the training data and got better results for all the fractional inputs than the previous models. Hence, the simple Logistic Regression model with flattened input was chosen as our best model for Dataset-2. The results are reported in Table 4.

Colab Notebook link- Click here

	Logis. Regression		**64-dim h-layer + LR**		**Logistic Regression**	
% of i/p	**Mean Pool**	**Max Pool**	**Mean Pool**	**Max Pool**	**PCA - 300 dim**	**Flattened i/p**
20	0.50	0.47	0.48	0.48	0.91	**0.97**
40	0.48	0.46	0.49	0.45	0.90	**0.98**
60	0.47	0.48	0.49	0.48	0.94	**0.97**
80	0.46	0.49	0.45	0.51	0.95	**0.99**
100	0.45	0.50	0.45	0.44	0.93	**0.99**

Table 4: Valid. accuracies after training on subsets of training inputs of the Deep-Feature Dataset

3.3 Text Sequence Dataset

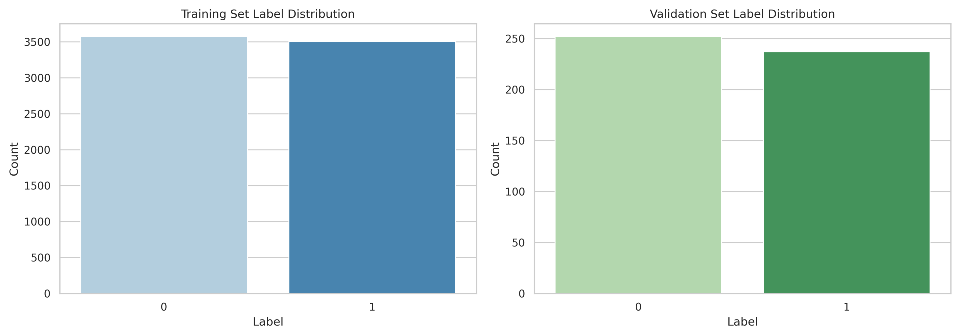
We have 7080 inputs in the training dataset consisting of 50-digits text sequences and their corresponding binary labels ({0, 1}).

Data pre-processing- During preprocessing, we observed that the classes were evenly distributed, fig. 2(a). We analyzed the distribution of individual digits and their labels, noting that '7' had consistently low frequency, followed by '8', '9', and '3', while '2' and '6' peaked, figs. 2(b) & 2(c). However, these findings were not useful for model performance as their labels remained balanced. We then explored consecutive digits with techniques like TF-IDF but again found no meaningful patterns. Despite testing multiple models on these insights, we ultimately abandoned this approach in favor of more effective/general vectorization techniques like one-hot encoding, numeric representation, and embeddings. To further understand the data, we tried Principle Component Analysis (PCA) [1] in 2D, fig. 3(a) and 3D, fig. 3(b) after converting the text sequences into numeric representation, to no one's surprise the points crumbled closely.

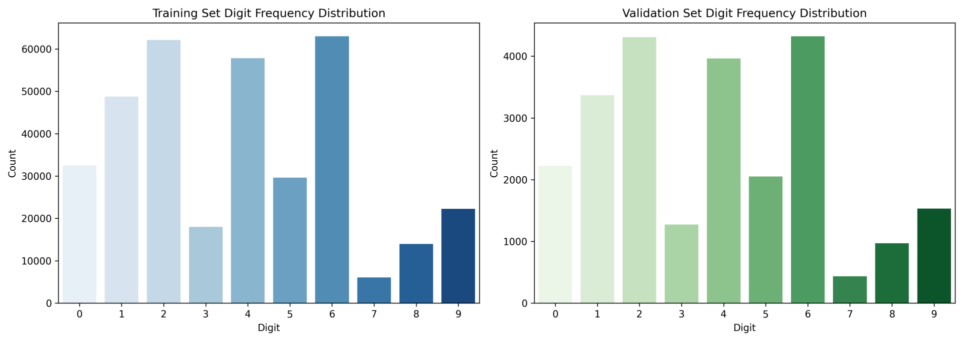
TF-IDF- (Term Frequency-Inverse Document Frequency) [3] is a statistical measure used to evaluate the relevance of a word in a document relative to a collection of documents (corpus). Term frequency (TF) calculates how often a term appears in a document, while inverse document frequency (IDF) reduces the weight of common terms across all documents. But, to our understanding the reason that it didn't work effectively for our dataset of digit-based text sequences because it is designed to capture word relevance in natural language, where terms have semantic meaning. In our case, the sequences consist of digits with no inherent meaning or contextual relationships between them.

One-hot encoding- One-hot encoding is a technique that converts categorical values into binary vectors, where each unique category is represented by a vector with all zeros except for a single 1 at the index corresponding to that category.

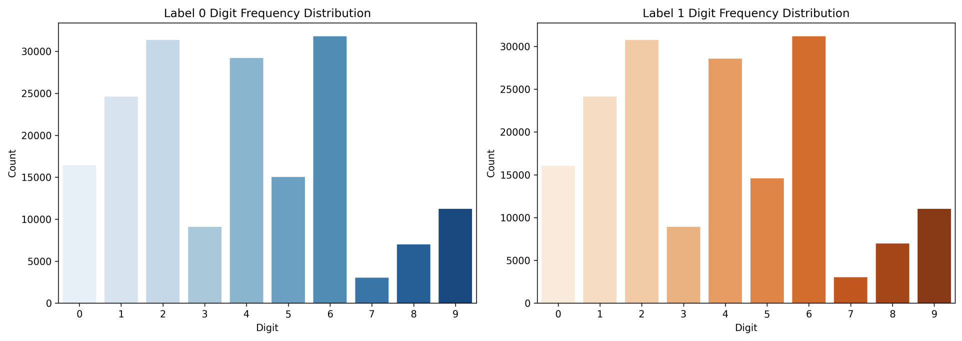
We trained various models the models listed in the table, 6, with incremental training datasets size. Multilayer perceptron was trained with two hidden layers of 100 and 50 neurons with ReLU and an output layer with sigmoid activation function with Stochastic Gradient Descent (SGD) and Binary Cross-Entropy Loss (BCELoss) for 500 epochs with learning rate of 0.005. Simple Neural Network consists of three fully connected layers with 64, 32, and 1 units, respectively. It uses ReLU activations, a dropout layer (with a rate of 0.5) for regularization, and a sigmoid activation for binary



((a)) Class distribution



((b)) Digit frequency distribution

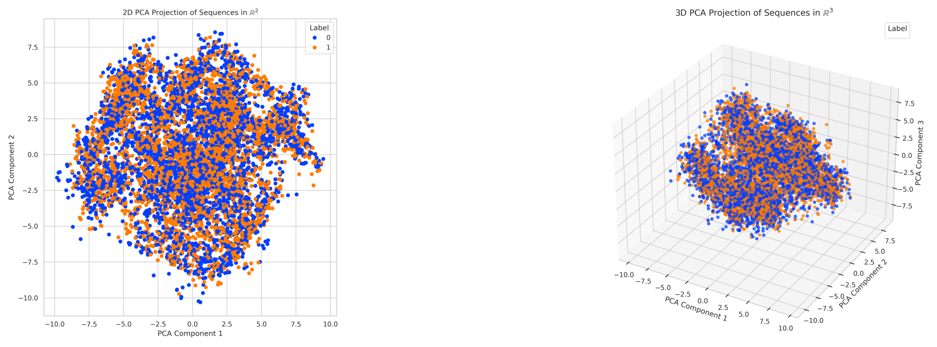


((c)) Digit-Label frequency distribution

Figure 2: Class Distribution, Digit distribution, and Digit-Label frequency distribution in text sequence dataset. These visualizations helped explore any potential patterns in the data. However, no useful associations between digit frequency and labels were found.

Model	**20% of i/p**	**40% of i/p**	**60% of i/p**	**80% of i/p**	**100% of i/p**
Logistic Regression	**0.52**	0.48	**0.52**	**0.52**	**0.52**
SVM (Linear)	0.51	0.48	0.51	0.51	0.51
SVM (Polynomial)	0.51	0.48	0.51	0.51	0.51
SVM (RBF)	0.51	0.48	0.51	0.51	0.51
Decision Tree	**0.52**	0.48	**0.52**	**0.52**	**0.52**
Random Forest	**0.52**	**0.52**	**0.52**	**0.52**	**0.52**
Naive Bayes	**0.52**	0.48	**0.52**	**0.52**	**0.52**
K-Nearest Neighbors	0.48	0.48	0.48	0.48	0.48
Single Layer Perceptron	**0.52**	0.48	0.48	**0.52**	**0.52**
Multilayer Perceptron	0.48	**0.52**	0.48	**0.52**	**0.52**
Simple Neural Network	**0.52**	0.48	**0.52**	**0.52**	**0.52**

Table 5: Validation Accuracy of Various Models Across Incremental Training Input Sizes using TF-IDF Encoding on Text Sequence Dataset



((a)) PCA in 2D ((b)) PCA in 3D

Figure 3: Principal Component Analysis (PCA) visualization in 2D and 3D for text sequence features.

classification. But the accuracy did not came out to be good enough. Also, the trainable parameters were off the charts for models like random forest and SVM with RBF kernel so we ditched the idea of using simple model but not without trying on different encodings.

Model	**20% of i/p**	**40% of i/p**	**60% of i/p**	**80% of i/p**	**100% of i/p**
Logistic Regression	0.63	**0.67**	**0.67**	**0.67**	0.65
SVM (Linear)	**0.64**	0.66	**0.67**	0.65	0.66
SVM (Polynomial)	0.56	0.60	0.63	0.64	0.65
SVM (RBF)	0.61	0.61	0.64	**0.67**	**0.67**
Decision Tree	0.53	0.55	0.54	0.54	0.55
Random Forest	0.56	0.58	0.60	0.63	0.64
Naive Bayes	0.58	0.60	0.58	0.60	0.61
K-Nearest Neighbors	0.53	0.53	0.53	0.51	0.55
Single Layer Perceptron	0.54	0.53	0.47	0.53	0.52
Multilayer Perceptron	0.52	0.48	0.50	0.52	0.51
Simple Neural Network	0.52	0.51	0.52	0.52	0.52

Table 6: Validation Accuracy of Various Models Across Incremental Training Input Sizes using One-Hot Encoding

Numeric representation- We transformed each of the fifty digits in the training sequences into their respective integer representations, creating a list of numerical values for model input, this had a clear benefit of reducing the number of trainable parameters by usually ten folds but the validation accuracy on aforementioned models only suffered with this, table 7.

Model	**20% of i/p**	**40% of i/p**	**60% of i/p**	**80% of i/p**	**100% of i/p**
Logistic Regression	0.53	0.54	0.52	0.54	0.54
SVM (Linear)	**0.55**	0.54	0.55	0.53	0.52
SVM (Polynomial)	0.51	0.57	0.57	0.57	0.58
SVM (RBF)	**0.55**	0.53	0.54	0.53	0.57
Decision Tree	0.50	0.53	0.54	0.53	0.57
Random Forest	**0.55**	**0.58**	**0.59**	**0.60**	**0.61**
Naive Bayes	0.54	0.55	0.54	0.54	0.56
K-Nearest Neighbors	0.49	0.53	0.52	0.53	0.54
Single Layer Perceptron	0.54	0.55	0.52	0.51	0.55
Multilayer Perceptron	0.54	0.52	0.53	0.46	0.54
Simple Neural Network	0.49	0.43	0.52	0.48	0.51

Table 7: Validation Accuracy of Various Models Across Incremental Training Input Sizes using Numeric Representation

LSTM model (Best)- The model employs a Long Short-Term Memory (LSTM) architecture with an embedding layer. The input layer takes in the numeric representation of the text sequence, followed by the embedding layer, which has a vocabulary size of 10 and an embedding dimension of 8. Next, we include an LSTM layer with 32 hidden dimensions. This layer processes the embedded sequences and captures temporal dependencies. Following the LSTM layer, we have a dropout layer with a rate of 0.6 to prevent overfitting. Subsequently, the model consists of four fully connected layers, where the first layer transforms the output from the LSTM to 16 units, the second layer reduces it to 8 units, the third layer further reduces it to 4 units, and the final layer outputs a single unit for binary classification. The output from this last layer undergoes a sigmoid activation function to produce a probability score. The model is trained using binary cross-entropy loss and optimized with the Adam optimizer, featuring a learning rate of 0.001 and a weight decay of 1e-5. The model is trained for 100 epochs and the final trainable parameter count of this model comes out to be only 6161, table 8.

Colab Notebook link- Click here

% of i/p	Training Accuracy	Validation Accuracy	Train Loss	Valid Loss
20%	0.76	0.66	0.51	0.68
40%	0.81	0.72	0.45	0.59
60%	0.50	0.52	0.69	0.69
80%	0.80	0.75	0.48	0.56
100%	**0.90**	**0.84**	**0.23**	**0.38**

Table 8: LSTM Model (Best Model) Performance on Incremental Input Sizes

3.4 Combined Dataset Model (Task 2)

We began our machine learning project by utilizing a majority voting technique on our previously developed models. The idea was to combine the outputs of these models, assuming that the collective decision would enhance the overall performance. However, this initial approach did not work as expected. The majority voting strategy, while often effective in ensemble methods, failed to improve accuracy or consistency in our case, leading us to reconsider our approach.

Next, we attempted a different strategy. Instead of relying on older models, we created new models tailored for each of the three datasets we were working with. These models were intentionally kept simple to avoid overfitting and to allow for easier interpretability. Once these models were developed, we merged their outputs using a more complex model, specifically a Deep Neural Network (DNN). Our expectation was that the DNN would capture intricate relationships between the outputs of the simpler models, improving the final performance. However, this approach also fell short of delivering the desired results. Despite the added complexity, the DNN did not significantly improve the model's overall accuracy or generalization across datasets.

Finally, in our last attempt, we returned to using our original models, one for each dataset, and combined their outputs using a simple Dense layer. To our surprise, this straightforward approach outperformed all previous methods. The simplicity of the Dense layer allowed for more efficient merging of the outputs without the unnecessary complexity introduced in earlier attempts. This final model not only achieved better performance but also demonstrated greater stability and robustness compared to the other models. The model is trained for 100 epochs and the total number of trainable parameters are 7518. The results are depicted in Table-9

% of i/p	Model 1 accuracy	Model 2 accuracy	Best Model accuracy
20%	0.79	0.85	**0.88**
40%	0.82	0.88	**0.92**
60%	0.83	0.88	**0.93**
80%	0.83	0.88	**0.95**
100%	0.87	0.90	**0.98**

Table 9: Accuracies of the three combined dataset models

Colab Notebook link (Best Model) Click here

4 Conclusion

In conclusion, our project highlighted the importance of balancing complexity with simplicity in machine learning architectures. For Emoticons dataset, a 20 dimensional embedding layer led to validation accuracies of more than 95%. After trying various dimensionality reduction techniques a simple flattening of inputs and Logistic Regression resulted in achieving the best validation accuracy for Deep Feature Dataset. The Text Sequence dataset proved to be the toughest to learn and finally a complex with LSTM, embedding layers and dropout gave the best accuracy of more than 85% For the combined dataset setting, While we initially believed that more complex methods would yield better results, the final solution, merging the original models with a simple Dense layer—proved to be the most effective. This iterative process of trial and error helped us identify the optimal model and underscored the value of simplicity in certain machine learning contexts.

References

[1] Pearson K. On lines and planes of closest fit to systems of points in space. *Phil. Mag. 2*, page 559–572, 1901.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[3] K. Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, page 559–572, 1972.