CS771 Assignment 1 Report

Group 26: ML-Mog

Group Details

Name	Roll Number	Email
Abhijeet Verma	210026	abhiteet21@iitk.ac.in
Aditya Saraf	210067	saditya21@iitk.ac.in
Chinmay Amrutkar	210286	chinmayma21@iitk.ac.in
Deependra Verma	210313	deependrav21@iitk.ac.in
Rohan Phad	210718	rohanpv21@iitk.ac.in
Sanyam Pasricha	210929	sanyamp21@iitk.ac.in

Emoticon Dataset

Dataset: The dataset contains emojis with each data point having the same number of emojis. To begin with, EDA was performed and insights were gathered to form the approach towards the problem.

Size of the training dataset	7080
Size of the validation dataset	489
Size of the test dataset	2232
Number of unique emojis in the train set	214
Length of each input	13

Data Preprocessing

Upon investigation from the EDA, it was observed that the emojis represent a string, and each unique emoji can be thought of as a character. Since machine learning models require us to deal with numbers, the emojis had to be converted to numbers before beginning the training process. The following two approaches were considered during training:

- Frequency count on each input
- One-hot encoding for each emoji in an input

Upon evaluating both methods, we made some observations and chose the most suitable model within the sample space.

Frequency Count

This approach involves forming a 214 (number of unique emojis) sized vector corresponding to each training input. Each emoji can then be mapped to a particular index, and the map is precomputed and stored throughout model training and evaluation. We iterated through each input and incremented the count of the corresponding index for all 13 emojis in a given input.

The advantage of this approach is that it results in a relatively small (214) sized input vector which makes training easy and fast, no additional preprocessing beyond this point is required.

Upon training a logistic regression model with this approach the validation accuracy came out to be a meagre 56%, which is just slightly better than a random guess considering binary classification.

This quickly led to the realisation that the spatial or sequential information related to these inputs is lost due to this method and therefore the accuracy might be less, as we are losing much of the information captured in the given input.

One-hot encoding for each emoji

In an attempt to capture all spatial information, we finally decided to use a one-hot encoding for each of the 13 inputs that would result in an input size of (13x214) for any given example. This can also be thought of being an image of size 13x214 with all pixels being either on or off. Further the input vector was flattened to yields a 2782 dimensionally vector for each input, flattening was necessary to ensure compatibility with Machine Learning API's like sklearn in this particular case. Further analysis was done using this pre-processing technique to compare different train set splits and also to evaluate different types of models.

Model Evaluation

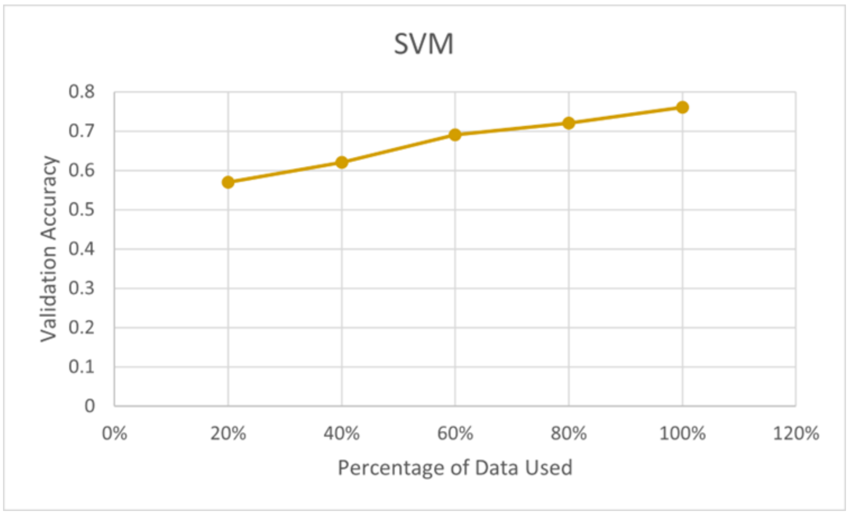
The following models were tried with all the different training data splits:

- Logistic Regression
- Decision Trees
- SVM

Let us know look at the accuracies given by each of these models, beginning with SVM, sklearn was used as an interface to train all of the three models.

Training Fraction	Accuracy
20%	0.57
40%	0.62
60%	0.69
80%	0.72
100%	0.76

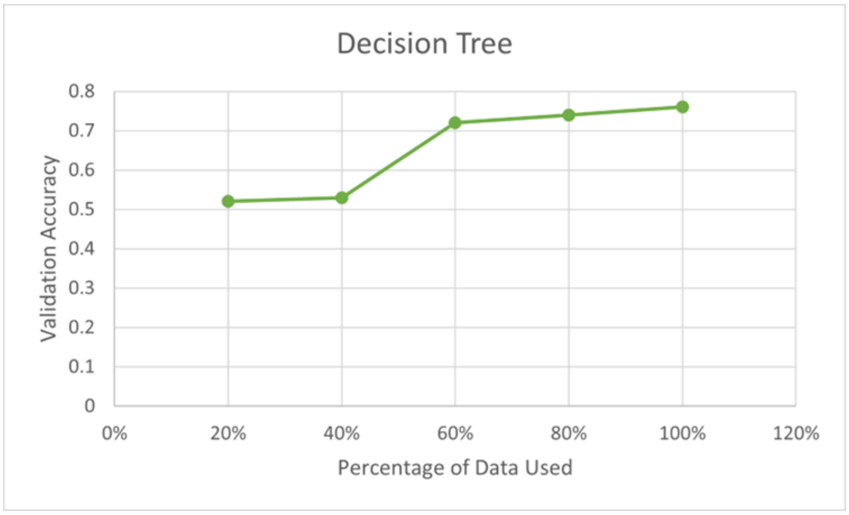
Images



Decision Tree

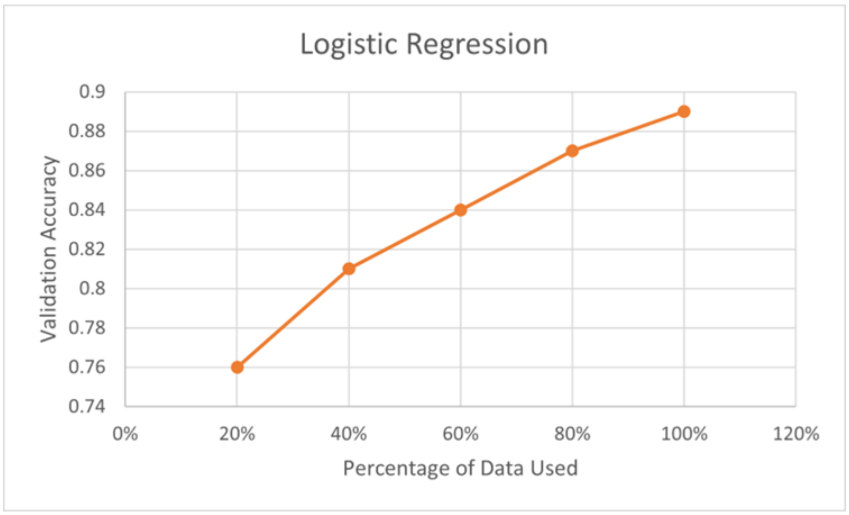
Training Fraction	Accuracy
20%	0.52
40%	0.53
60%	0.72
80%	0.74

100%	0.76



Logistic Regression

Training Fraction	Accuracy
20%	0.76
40%	0.81
60%	0.84
80%	0.87
100%	0.89



As observed logistic regression yields by-far the best accuracy amongst the three models, also the accuracy keeps increasing with the increase in training data, to avoid over-fitting while making an attempt to capture all patterns, it seems fit to use 80% of the training data. Additionally, it is observed that the model has an impressive accuracy of 97.7% on the training set.

Challenges

While making the predictions on the given test set, a few data points were observed which had emojis which were never seen in the training set before, to deal with the situation while maintaining the procedure of the proposed pre-processing it is decided to leave all elements corresponding to that particular emoji as zeros in the input.

Deepfeat Model: Data Preprocessing

During the exploratory data analysis (EDA), it is observed that the dataset consists of inputs that are represented as matrices. We flattened the matrices into a vector for each input. This approach allows us to preserve the overall structure of the data while ensuring compatibility with standard libraries like scikit-learn. The following preprocessing approaches were considered for training:

Flattening the Dataset

The matrix inputs were flattened into a one-dimensional vector for each sample. Specifically, the dataset consists of matrices, where each training point was originally represented in a 2D format. For model training, we flattened each matrix into a single vector.

In this approach, the data retains its original structure, just transformed into a flattened vector. Each input is now represented by a 1D vector with the same length for all samples, making it easier to work with standard machine learning libraries.

However, one trade-off is the potential loss of spatial patterns present in the original matrix format.

Averaging the Dataset

This approach involves taking the input matrix and averaging its values across the smaller dimension, effectively reducing the dimensionality of the input data. By doing so, the dataset's dimensionality was reduced. The advantage of this method is that it reduces the computational complexity, enabling faster model training while still maintaining important features of the original data.

Upon initial evaluation it was observed that the model had accuracies around 55%, therefore making it unsuitable, also revealing the information spread in the data, as in that all values were contributing to the information and were not redundant which is often the case in multiple datasets.

Model Evaluation

The following models were tried with all the different training data splits as instructed in the problem statement:

1. Logistic Regression (Average Data) 2. Logistic Regression (Flatten Data) 3. Perceptron (Flattened Data) Let us look at the accuracies given by each of these models, beginning with Logistic Regression, also sklearn was used to train all of the three models.

Logistic Regression (Average Data)

Training Fraction	**Accuracy**
20%	0.51
40%	0.50
60%	0.48
80%	0.45
100%	0.46

Logistic Regression (Flatten Data)

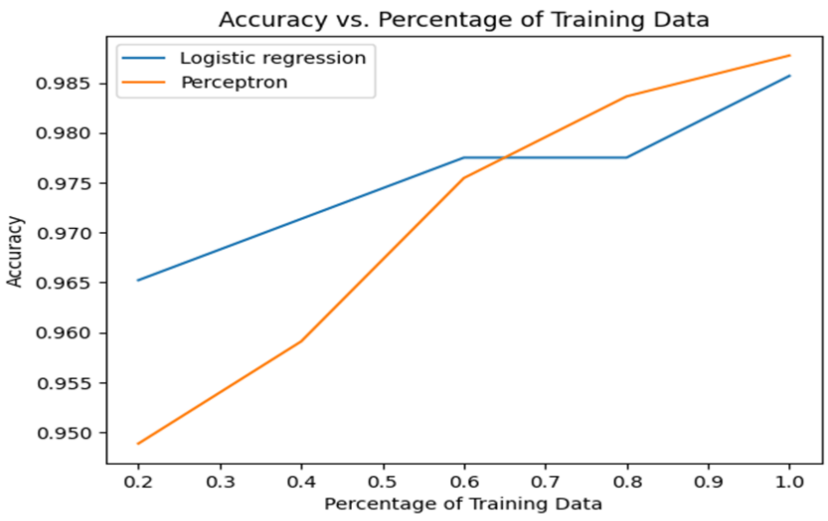
Training Fraction	**Accuracy**
20%	0.96

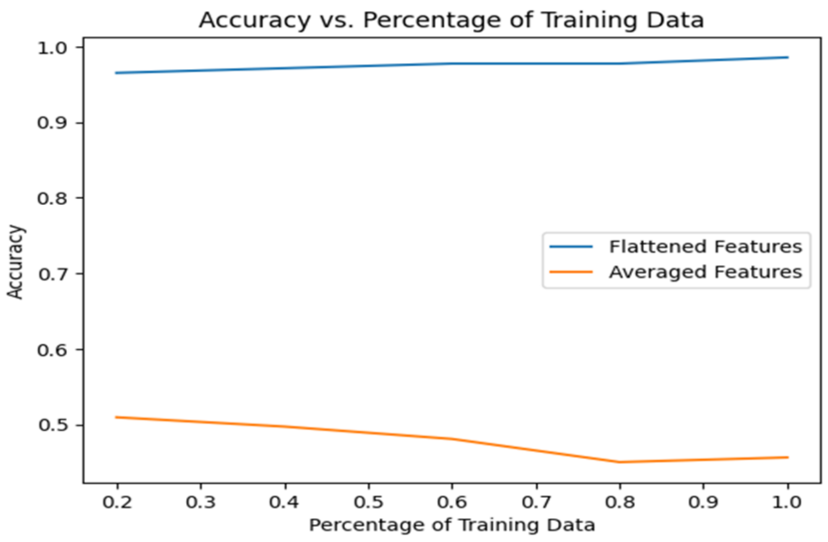
40%	0.97
60%	0.98
80%	0.98
100%	0.98

Perceptron

Training Fraction	Accuracy
20%	0.94
40%	0.96
60%	0.97
80%	0.98
100%	0.98

Images





As observed logistic regression yields by-far the best accuracy amongst the three models, also the accuracy keeps increasing with the increase in training data, to avoid over-fitting while making an attempt to capture all patterns, it seems fit to use 80% of the training data.

Additionally, it is observed that the model has an impressive accuracy of 98.77% on the training set.

Text-seq Dataset

Dataset: The dataset contains text sequences as input where each input is a 50-length string consisting of digits from 0-9. To begin with, EDA was performed and insights were gathered to form the approach towards the problem.

Size of the training dataset	7080
Size of the validation dataset	489
Size of the test dataset	2232

It was observed from the data that the digits had a numeric and as well as positional significance, that is both their numerical value and position are important. One more thing that was observed was that there is a countable set of 3-digit, 4-digit and 5-digit numbers that are recurring in all the input strings at different locations. Also the zeros at the beginning of the string are just padding to make the input lengths equal.

Data Preprocessing

The input data is in the form of strings of length 50 consisting of digits. From the above analysis it can be seen each digit or a set of digits have to be considered as a categorical data and not as a real valued data. Also, the zeros at the beginning of dataset have no significance. Thus, we have followed the approach of one-hot encoding the data. The one-hot encoding is done in 2 different ways and compared:

- Encoding each digit character of the input string as a length 10 vector.
- Encoding 2 consecutive digit characters as a length 100 vector.

Further the input vector was flattened to yields a 500- and 2500-dimensional vector for each input, flattening was necessary to ensure compatibility with Machine Learning API's like sklearn in this particular case.

For both encoding the model is trained once encoding the padding zeros as well and then not considering them by encoding them as zero vector. Trigram and further one-hot encoding is not done as that exceeds the trainable parameter's limit.

A CatBoost model is also trained where the data is not one-hot encoded. Each input string is converted to a 50 length input vector with the features as the individual digits. Here also bi-gram representation is also tried, that is each input is converted to a 25 length vector of 2 digit numbers.

Model Evaluation

The following models were tried with all the different training data splits:

- Logistic Regression
- SVM
- XGBoost
- CatBoost

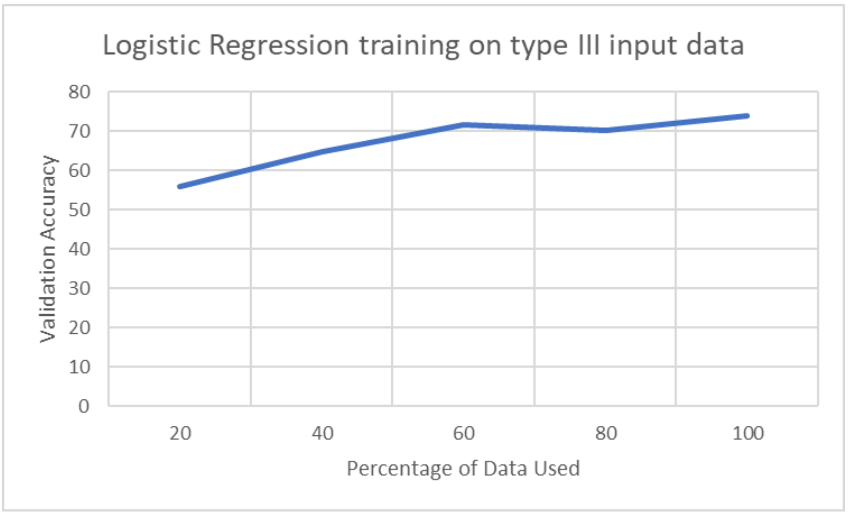
Logistic Regression model is trained on the one-hot encodings of the data. It yields the following results. We also trained a logistic regression model with L1 regularizer as the input data is sparse and thus many features might be less important and L1 regularizer will reduce their weight magnitude. But the accuracy was lower than that achieved by L2 regularizer, so the L2 regularizer was used.

The tolerance and regularization hyperparameter was fine-tuned through cross-validation and the values of 0.0005 and 1 was used.

Logistic Regression Results

Training Fraction	Encoding zero padding (I)	Neglecting zero padding (II)	Encoding zero padding (III)	Neglecting zero padding (IV)
20%	63.19%	63.60%	60.12%	60.33%
40%	66.46%	66.46%	66.67%	67.08%
60%	66.05%	65.64%	68.71%	69.12%
80%	65.64%	66.05%	69.33%	69.12%
100%	65.24%	65.64%	71.37%	70.96%

Images



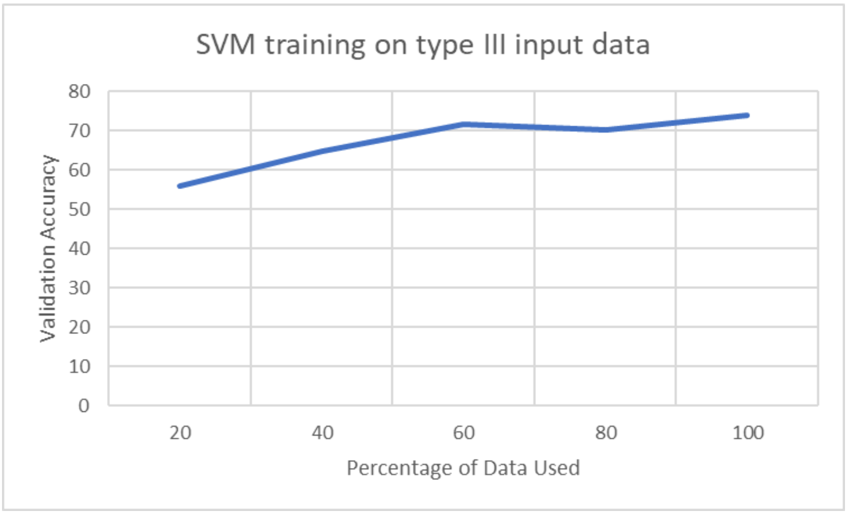
SVM Results

SVM model was again trained on the 2 different encodings considering the padding zeros and not considering them.

The soft-margin trade-off hyperparameter C was fine-tuned. An ideal value of 10 was chosen through cross-validation. The following results were obtained.

Training Fraction	Encoding zero padding (I)	Neglecting zero padding (II)	Encoding zero padding (III)	Neglecting zero padding (IV)
20%	63.60%	62.78%	59.51%	59.30%
40%	65.03%	65.24%	65.64%	65.64%
60%	66.67%	66.26%	67.28%	67.28%
80%	65.85%	66.26%	70.14%	70.35%
100%	65.24%	65.24%	70.76%	70.96%

Images

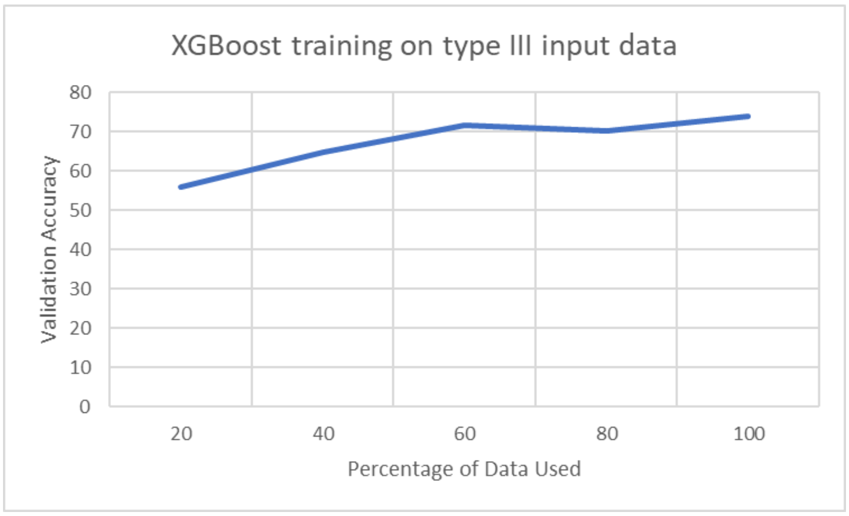


XGBoost Results

Since we have a categorical input data, it was a good idea to train a decision tree-based model. The xgboost model is an ensemble and utilizes the technique of boosting. The hyperparameters ‘number of estimators’ and ‘max depth of tree’ were fine-tuned to get values of 2000 and 7 respectively. The following results were obtained.

Training Fraction	**Encoding zero padding (I)**	**Neglecting zero padding (II)**	**Encoding zero padding (III)**	**Neglecting zero padding (IV)**
20%	60.12%	59.71%	55.83%	54.60%
40%	63.39%	64.21%	64.83%	64.83%
60%	67.69%	66.46%	71.78%	70.35%
80%	66.26%	67.28%	70.35%	69.12%
100%	68.10%	69.12%	74.03%	73.62%

Images



Observations

Among the 3 models logistic regression is the most appropriate model. 1. The accuracy of Logistic regression, SVM and XGBoost on the training data are about 86%, 87% and 99% respectively. Thus, it clearly shows that XGBoost is the most overfit model. Even after fine-tuning the parameters, although there is a bit decrease in overfitting, but still it is quite high as compared to other models for similar or lower validation set accuracy. Logistic regression is the least overfit model and thus is the best choice among the 3 models.

2. Logistic regression has a good accuracy on even lower amounts of data. For other models the accuracy on lower fractions of data is quite low as compared to Logistic regression with the lowest being for XGBoost.

It can also be observed bi-gram encoding of data with one-hot encoding of padding zeros gives the best accuracy across the three models.

The highest accuracy achieved is 74.03% for the xgboost model with the bi-gram encoding(including padding zeros).

CatBoost

The CatBoost model is the ideal model to train a boosting model on a categorical kind of data. Here the model was trained on 1-gram and bi-gram and tri-gram list vector representation of the input string, that is a list of each character and list of strings taking 2 characters at a time.

The model accuracy was very low around 51% which is quite low for a binary classification model.

We also tried putting 1-gram, bigram and trigram of the string in one input vector and then running the model, but still the accuracy didn't change.

Other Models

Some other basic models were also tested like k-Nearest Neighbours, Decision Trees, Random Forest and stacking classifier of XGBoost and SVM. But the accuracies achieved were about 60-65

1 Combined Model:

Data Processing:

The best accuracy by far was obtained on the deep-feat dataset, which was an almost perfect 98.7%, but the number of parameters were more than 9800 which is very close to the limit of 10,000 parameters, thus preventing us from using other datasets as additional features to this.

Therefore, we tried using the emoticon and text-seq datasets together, combining both their inputs together and training a new model.

Model Evaluation:

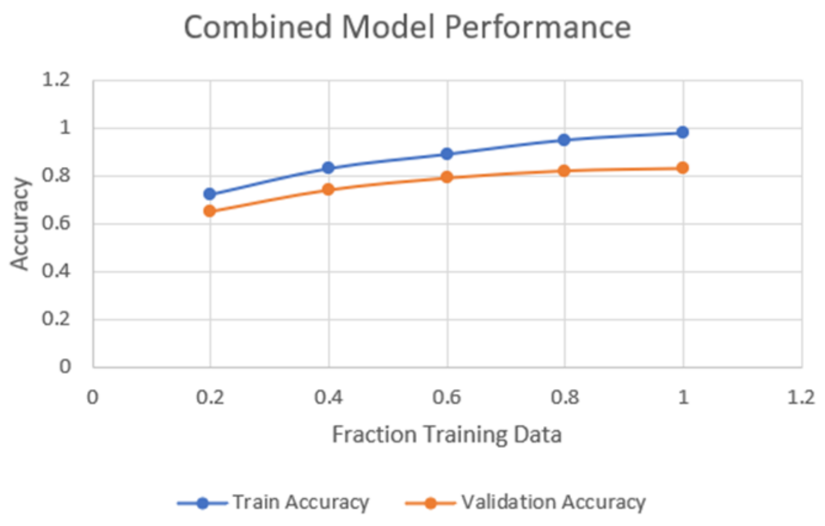
Across all the three individual datasets it is observed that the only consistent best performing model has been Logistic Regression, therefore we chose to use Logistic Regression for the combined model as well as the model seems to suit the data pretty well.

As instructed, we trained logistic regression across all five data splits from 20% to 100% and the results were as follows, increasing continuously with and increase in training data size.

Training Fraction	Train Accuracy	Validation Accuracy
0.2	0.72	0.65
0.4	0.83	0.74
0.6	0.89	0.79
0.8	0.95	0.82
1	0.98	0.83

Table 13: Training and Validation Accuracy for different Training Fractions

Images



The best accuracy the model could give is around 83%. The ideal training data split is again 80%, giving similar results as 100% data while still keeping some part of the dataset as unseen.

Best Model:

The Best Model that we could train is a **Logistic Regression with 80% training data on the deep-feat dataset**. The accuracy was upwards of **98%** which beats all models including the combined ones.