University of Illinois Urbana-Champaign

Department of Statistics

Classification of hand gesture labels into letters in English Language

Group Members:
Lead: Sharvi Tomar (stomar2)
Shubahm Mehta (mehta45)
Anushree Vilas Pimpalkar (avp4)

A project report submitted for the course of

STAT-432 Basics of Statistical Learning

Contents

Chapter 1: Introduction

The Sign language is a visual language expressed by facial expression as well as the hand motions, used by the people with speech and hearing disabilities to communicate with others, and to express their views and opinions. According to the World Federation of the Deaf, there are 70 million people in the world who use sign language.

1.1 Goal

In this project, our objective is to correctly identify American Sign Language (ASL) signs that correspond to the hand gestures represented by English alphabets (A-Z, excluding J and Z, as they require motion). The project also compares and contrasts the performance of different machine learning models for the given multi-label classification task.

Figure 1.1: American Sign Language signs

1.2 Approach

The static sign language data for our project was in the form of images. Different machine learning algorithms like Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis(QDA), Naïve Bayes, K-Nearest Neighbours(KNN), Support Vector Machine(SVM), Random Forest(RF), Logistic regression and eXtreme Gradient Boosting(XGBoost) were implemented to identify the signs represented by each of these images. Data was feature engineered to get reduced dimensions using Principal Component Analysis(PCA) and Linear Discriminant Analysis(LDA) as dimensionality reduction techniques. Results obtained using the reduced data and the original data were compared. While understanding and interpreting the results obtained with the models, we came up with a different approach of bucketing and implementing level-wise classification to improve the accuracy of this multi-class classification problem.

1.3 Conclusion

The proposed approach of combining similar looking hand gesture labels into a combined label and then classifying them back into individual labels from those combined labels reduced the mis-classification rates of similar hand gesture labels. This was achieved by selecting the best model at every level, which increased the accuracy from 83% using the best performing model (KNN) to 84.3% with proposed approach on PCA-reduced data.

Chapter 2: Literature review

In this section, we have discussed related works on the American Sign language dataset considering vision-based approaches.

Shin et al. [SS21] used hand images from a web camera to conduct research to recognize American Sign Language labels. Instead of only using the images, the authors predicted the coordinates of the hand joints, treating them as the features based on the distance between the joints and angle between the direction of two joints and the X, Y, Z axis for recognition. They used support vector machine (SVM) and light gradient boosting machine (GBM) on the ASL Alphabet dataset, the Massey dataset, and the Finger Spelling A dataset. This resulted in 98.45 % for Finger Spelling A dataset, 99.39 % accuracy for the Massey dataset and 87.60 % for the ASL Alphabet dataset. Using the coordinates as features, they obtained better results than using just the images.

Garcia et al. [GV16] implemented American Sign Language (ASL) in real time to translate a video of a user's signs into text using Convolutional Neural Network (CNN) by utilizing GoogLeNet architecture trained on the ILSVRC2012 dataset. They created a pipeline, which uses a web application to take the video of a user signing a word as an input. Using the individual frames of the video, they generated the letter probabilities for each letter using CNN. The frames were then grouped based on the letter index that each frame is most likely to represent. At the end they were to generate predicted words using a language model and consistently correctly classify alphabets a-e with first-time users and a-k in most of the cases. They attained a validation accuracy of nearly 98 % for a-e, 74 % for a-k ((excluding j) and 66 % for a-y (excluding j). The validation accuracies that were observed during training were not directly reproducible upon testing on the web application because of lack of variation in the data. The authors believed that this difference could be reduced by considering additional data in different environmental conditions.

Dhruv Rathi [Rat18] developed a system based on Mobile platforms that uses live video frames to recognize and extract labels of the American Sign Language(ASL). The author used Transfer Learning in order to have a robust and flexible system for both small or large datasets. The classification accuracy obtained was 95.03 %.

Mohit Patil et. al. [MP20] introduced a grid based recognition system for recognition of gestures and poses of Indian Sign Language (ISL) which provides both real time detection and precision. The system comprises an Android smartphone camera to capture hand poses and gestures, and a server to process the frames received from the smartphone camera. Approaches like segmentation, face detection, object stabilization and skin color were used to effectively classify all 33 hand poses in Indian Sign Language. The approach uses a k-NN model to classify each hand pose an HMM chain for each gesture. The recognition system could predict all 28 (8 digits and 20 letters) hand poses in Indian Sign Language with 96.4 % accuracy and 10 gestures with 98.23 % accuracy on an average. The presented approach can be further extended to other sign languages, like American Sign Language (ASL).

Chapter 3: Data Processing

3.1 Dataset Source

The data set is taken from the MNIST database (Modified National Institute of Standards and Technology), which is a large database of handwritten digits that is commonly used for training various image processing systems.

3.2 Data Dimensions & Label Mapping

Training Data: 27455 x 785 | Testing Data: 7172 x 785 Each training and test case represents a label (0-24) as a one-to-one map for each alphabetic letter A-Z (excluding J and Z)

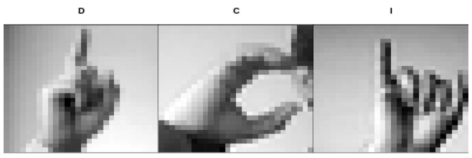

Figure 3.1: Sample Grayscale Images

A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
0	1	2	3	4	5	6	7	8	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

Figure 3.2: Mapping of labels to A-Z(excluding J,Z)

3.3 Checking for unusual observations

Missing values, Pixel with negative values & constant intensity

```
%%R
negative_val_cols_train <- list()
negative_val_cols_test <- list()
for(i in 2:785){
    if(min(train[,i])<0){
        negative_val_cols_train = append(negative_val_cols_train,i)
    }
    if(min(test[,i])<0){
        negative_val_cols_test = append(negative_val_cols_test,i)
    }
}
print(c(length(negative_val_cols_train),length(negative_val_cols_train)))
```

```
[1] 0 0
```

Figure 3.3: Negative-valued pixel

```
%%R
constant_val_cols_train <- list()
constant_val_cols_test <- list()
for(i in 2:785){
    if(min(train[,i])==max(train[,i])){
        constant_val_cols_train = append(constant_val_cols_train,i)
    }
    if(min(test[,i])==max(test[,i])){
        constant_val_cols_test = append(constant_val_cols_test,i)
    }
}
print(c(length(constant_val_cols_train),length(constant_val_cols_test)))
```

```
[1] 0 0
```

Figure 3.4: Constant-valued pixel

The train as well as test data does not have any missing/NA values so we not have to impute or handle missing observation. The data also does not have any pixel value as negative which could have been a reporting error or incorrect observation. We also checked if any pixel has constant values throughout the data as this pixel column could then have been removed since it does not provide any useful information. However, the data is free of any such pixel.

3.4 Data Distribution & Evaluation Metric

We next checked if the dataset is balanced or not. In case of unbalanced data, we would require to perform sampling and select appropriate evaluation metric. However, the data seemed balance and hence, we selected 'Accuracy' as evaluation metric for the task.

$$Accuracy = \frac{\text{Number of correct observations}}{\text{Total Number of observations}} \tag{3.1}$$

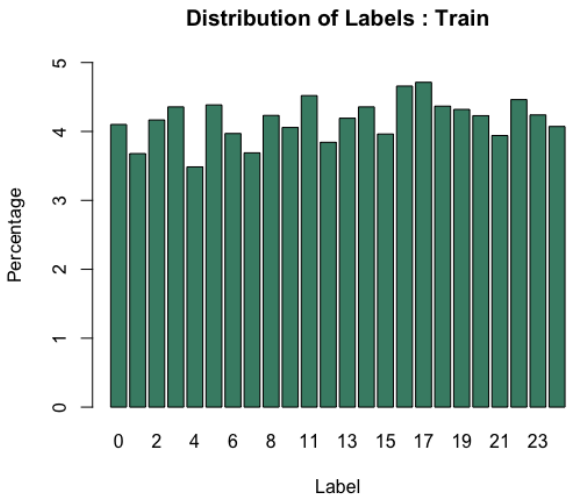

Figure 3.5: Train: Labels Distribution

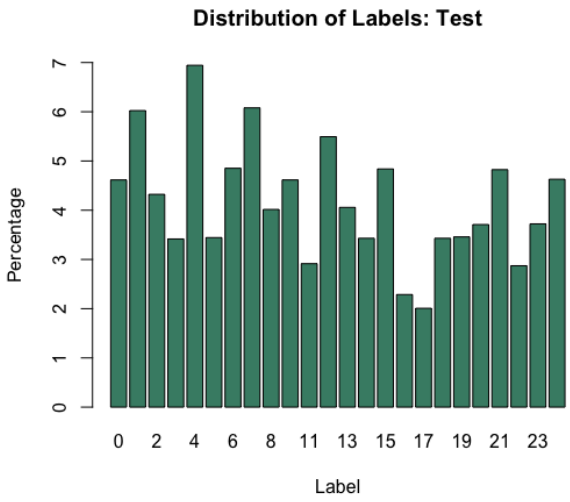

Figure 3.6: Test: Labels Distribution

3.5 Dimensionality Reduction

3.5.1 Principle Component Analysis

Principal Component Analysis (PCA) is an "unsupervised" learning algorithm that works by identifying the hyperplane that is nearest to the data and then projecting the data on that hyperplane along with retaining most of the variation. With 784 features in our original data, dimensionality reduction will help reduce the number of features to the most relevant ones, making the data easier to explore, visualize and analyze.

Each Principal Component is a linear combination of original variables pixel1, pixel2, ... pixel784. The variation explained by the Principal Components decreases with every new PC. The graph on the left below, represents the first 2 Principal Components, where 35.7 % of the variation in the dataset is explained by the 1st Principal Component. The 2nd PC which is orthogonal to the 1st PC explains 8.3 % of the variation in the dataset. The graph on the left represents the cumulative variance explained by all Principal Components. We find that the first 100 PCs explain more than 94 % of the variation. Therefore, now we will consider the first 100 Principal components as the predictors to predict our labels, instead of the initial 784 columns. Hence, we have successfully reduced our dimensions from 784 to 100.

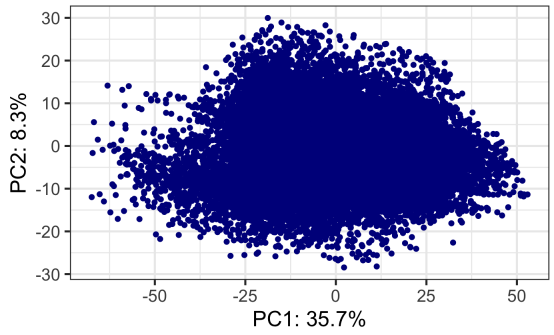

Figure 3.7: Principle Components

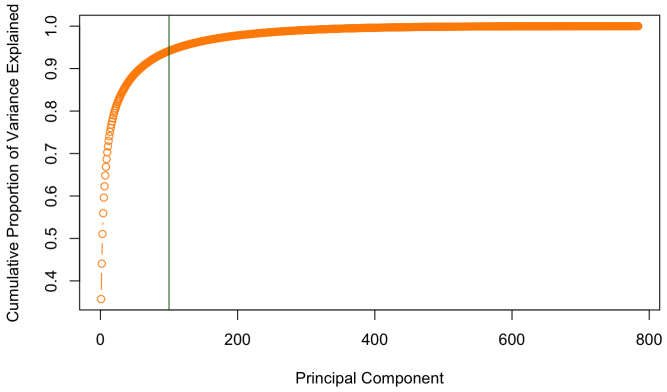

Figure 3.8: Cumulative Variation Explained

label	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	...	PC91	PC92	PC93	PC94
3	4.556731	-0.764484	-7.749799	3.210755	1.457682	-7.899003	-0.578572	-6.384985	1.542336	...	-0.513050	0.480673	0.286046	-0.469314
6	6.605218	4.609541	3.231124	4.628708	9.867037	-2.513497	-0.478265	1.066368	-1.357196	...	0.913640	1.145643	1.048088	-0.273757
2	-0.530982	20.312834	-6.114354	4.063660	11.600203	2.622752	-7.895567	5.347095	-4.596339	...	-0.689630	-0.016129	-0.196506	0.672759
2	-9.808229	20.537462	3.690911	9.295835	14.625165	2.347955	16.041041	5.254205	0.960582	...	-1.937599	0.674054	-0.473524	0.399404

Figure 3.9: Resulting Principal Components

3.5.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) can be used both as a classification algorithm and dimensional reduction technique. This technique tries to find the directions/dimensions that maximize the separation between multiple classes (shown in figure 3.10 below), and minimize the within class variance after projection.

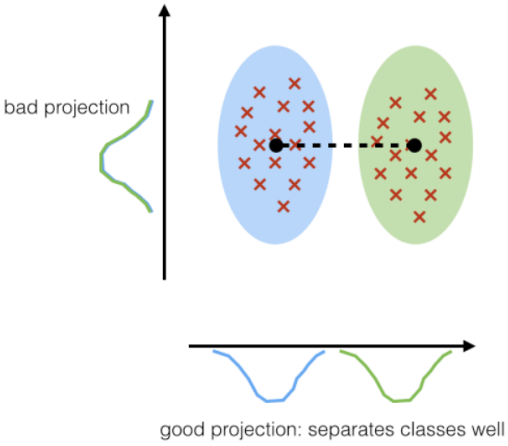

Figure 3.10: LDA: Projection of 2 classes

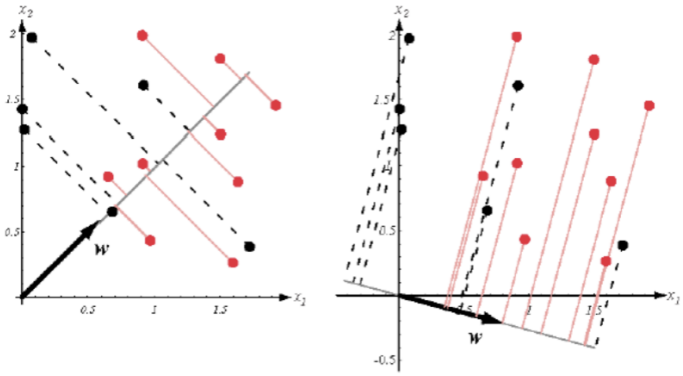

Figure 3.11: Different projection options

In figure 3.11, data from 2 classes (red and black) is projected onto a line. Since our goal is to prefer a projection where the projected data from two classes are as far as possible, the projection on the right is preferred. Some of the important aspects of LDA dimension reduction technique:

- LDA can reduce the number of dimensions to C-1, where C is the number of classes. For our data, we had 23 new LD's (LD1, LD2. . . .,LD23) as we had 24 different classes.
- Each LD is a linear combination of the original dimensions.
- Each LD contains some variation of the data. Figure 3.13 shows the proportion of variation that each LD contains for the training data.

```
Proportion of trace:
   LD1    LD2    LD3    LD4    LD5    LD6    LD7    LD8    LD9   LD10   LD11   LD12   LD13   LD14   LD15   LD16   LD17   LD18   LD19   LD20   LD21   LD22   LD23
0.1400 0.1230 0.0989 0.0709 0.0652 0.0515 0.0435 0.0417 0.0402 0.0380 0.0333 0.0329 0.0286 0.0256 0.0246 0.0226 0.0213 0.0195 0.0187 0.0179 0.0156 0.0144 0.0121
```

Figure 3.12: Proportion of Trace

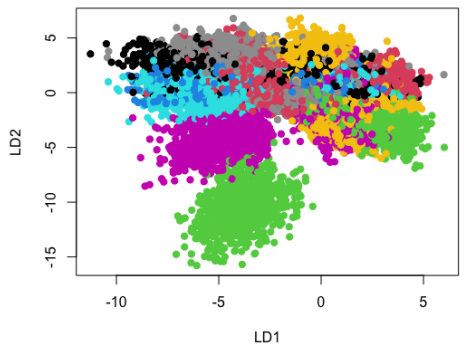

Figure 3.13: Train: Projection of 24 classes

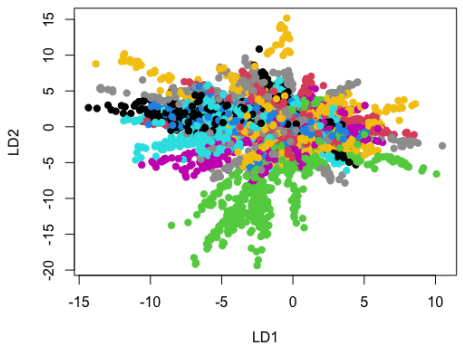

Figure 3.14: Test: Projection of 24 classes

In figure 3.13, LDA gives nice clusters for 24 different labels on the training data when projected on LD1 and LD2. On test data, we obtain a different separation for the different labels using LDA can be seen from the figure 3.14. This is because in high dimensions (N»1), it is difficult to obtain a precise estimate of within class covariance matrix, and one needs a lot more than N data points for getting a good estimate. Thus, when we have very high number of dimensions (like 784 in our data); the within class covariance matrix is almost singular and LDA tends to overfit with bad performance on test. This was validated as training accuracy was extremely high and testing accuracy was very low for every machine learning algorithm that we implemented on LDA-reduced data.

Chapter 4: Modeling

4.1 SVM

Support vector machine is a supervised machine learning algorithm, that can be used for regression, classification and also to detect any outlier in the dataset. It is one of the most widely used classification model, where we plot each data point in an n-dimensional space (n = no of features), and create a hyper-plane to best separate the data points as per the labels.

4.2 KNN

K nearest neighbor is a non-parametric supervised machine learning algorithm which can be used for both regression and classification problems. This algorithm perform a local averaging technique to estimate a label using similarities, by using observations closer to it. Here, we have used the cross validation technique to get the optimal number of neighbours in order to get minimum classification error. The figure at left shows the k vs classification error plot on the complete original data, while the left plot represents the same for PCA-reduced data. We find that they both result in k=1 as the optimal k value.

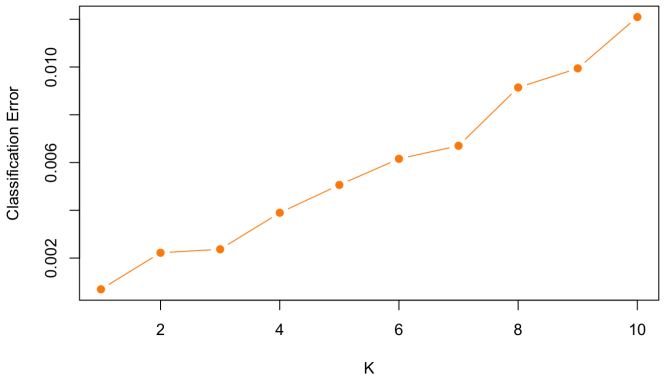

Figure 4.1: Original Data:k tuning

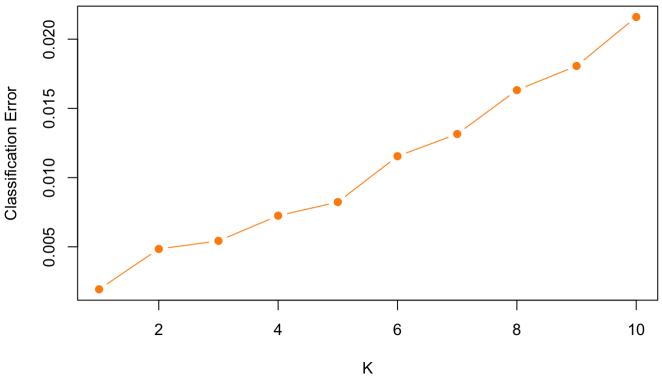

Figure 4.2: PCA-reduced Data:k tuning

4.3 Naive Bayes

This is a classification algorithm based on the bayes theorem. Naïve bayes classifier assumes that there is conditional independence between every pair of features given the value of the class variable.

4.4 LDA

This classification algorithm generates linear decision boundary. It uses bayes rule and fits a gaussian density to each class, with the assumption that all classes share the same covariance matrix. This method, as explained earlier can also be used as a dimension reduction technique.

4.5 QDA

QDA is a classification algorithm which generates quadratic decision boundary. This algorithm uses bayes rule and fits a gaussian density to each class, assuming that each class has its own covariance matrix.

4.6 Random Forest

A random forest algorithm trains many decision trees on various sub-samples of the data (creating a forest) and use them as classifier. It uses averaging to improve the predictive accuracy and control over-fitting. Important hyper-parameters like node size and number of features were tuned by splitting the training data into train and validation.

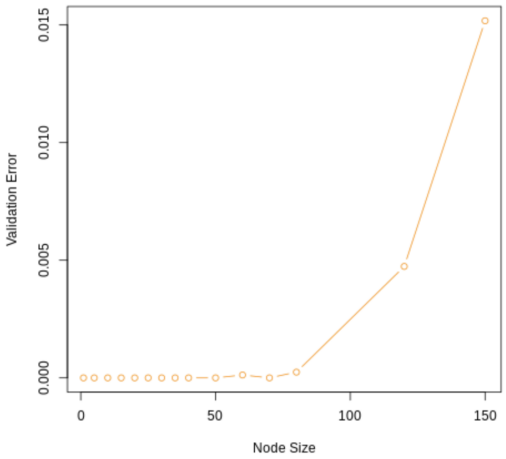

Figure 4.3: Validation Error vs Node Size

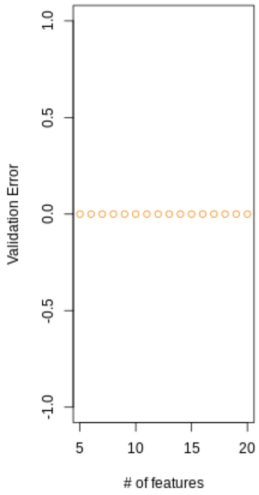

Figure 4.4: Validation error vs # features

As shown in figure 4.3, validation error remains zero as we change the number of features from 5 to 20. We selected 10 features (square root of number of features) in the final random forest in the PCA-reduced data. Similarly, 10 node size was selected for the final Random Forest model as validation error remained at zero for node size between 0 to 50. Thus, 10 was chosen to have both good generalization and avoid over-fitting.

4.7 XGBoost

A good model is the one with great generalization as well as prediction accuracy. Random Forest brings generalization by bootstraped aggregation while boosting brings greater prediction accuracy by sequentially building the trees on residuals of the prior trees. XGBoost is an efficient implementation of the gradient boosting algorithm designed for speedy computation in case of large and complex data.

4.8 Combined Results

4.8.1 Accuracies

Model	Original Data	PCA-reduced Data	LDA-reduced Data
Naive Bayes	38.97 %	62.28 %	43.84 %
LDA	43.26 %	62.07 %	43.26 %
QDA	65.47 %	71.5 %	37.4 %
SVM	85.05 %	77.69 %	40.95 %
KNN	81.04 %	83 %	46.44 %
Random Forest	81.58 %	81.34 %	39.96 %
XGBoost	76.56 %	74.27 %	30.1 %

- Naïve Bayes gives low accuracy on original data. It assumes that features are independent, which is not true in our case. For PCA reduced data, the independent assumption is true, since all principal components are orthogonal/independent. Thus, there is a huge jump in accuracy for Naïve Bayes model with PCA data.
- For other models as well, implementing PCA improved the accuracy due to reduction in noise/correlation.
- LDA dimension reduction resulted in lower test accuracy for every model because of overfitting as explained in dimension reduction section.
- LDA classifier produces linear decision boundaries resulting in lower accuracy as compared to QDA/KNN/SVM/XGBoost/Random Forest which can produce quadric/non-linear decision boundaries.
- KNN, XGBoost, SVM and Random Forest produces better accuracy among these 7 models for the PCA-reduced test data

4.8.2 Mis-classification of labels

	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
A	331	0	0	0	0	0	0	0	19	0	0	18	22	0	0	12	0	0	0	0	0	0	0	0
B	0	384	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C	0	0	309	0	0	2	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0
D	0	9	0	226	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	35	0	15	0	0
E	0	0	0	0	465	0	0	0	0	0	0	13	1	0	0	0	0	18	0	0	0	0	0	0
F	0	0	0	0	0	226	0	0	0	0	0	0	1	1	0	0	0	0	0	0	18	0	0	0
G	0	0	0	0	0	0	277	20	1	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	1	413	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	2	0	0	190	0	0	7	0	0	0	0	0	2	0	0	0	0	0	0
K	0	38	0	0	0	0	0	0	0	209	0	0	0	1	0	0	3	0	0	26	17	26	0	0
L	0	0	1	0	0	17	0	0	1	2	209	0	0	0	0	0	10	0	13	2	0	0	0	0
M	0	0	0	0	0	0	0	0	13	0	0	269	6	4	0	0	0	46	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	1	0	0	22	211	4	0	0	0	2	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	10	215	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	345	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	21	0	1	0	0	0	21	1	0	152	0	3	0	0	0	0	0	0
R	0	0	0	17	0	0	0	0	15	15	0	0	0	0	0	0	90	0	0	20	19	26	0	36
S	0	0	0	0	33	0	0	0	4	22	0	65	9	0	0	0	3	175	0	0	0	0	0	0
T	0	0	0	0	0	0	49	3	0	0	0	0	2	4	0	0	11	0	202	0	2	0	21	1
U	0	0	0	0	0	0	0	0	0	22	0	0	5	1	0	0	27	0	0	174	27	21	0	8
V	0	0	0	1	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0	4	152	5	0	21
W	0	1	0	0	0	0	0	0	0	19	0	0	0	0	0	0	0	0	0	4	108	113	0	15
X	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2	0	0	0	33	0	3	0	246	0
Y	0	0	0	0	0	0	0	0	27	9	0	0	0	0	0	0	0	0	0	0	0	0	0	251

Figure 4.5: Random Forest Confusion Matrix on PCA-Reduced Test Data

- In the figure 4.5, columns represent the Actual labels and rows represents the Predicted labels. It can be observed that label M was 18 times getting mis-classified as A,13 times as E, 22 times as N and 65 as S.
- Similar looking labels were getting misclassified. A,E,M,N,S ; G,H,T; U,R,D were getting mis-classified within each other.
- This served as motivation to group few labels together so that they don't get confused with other labels.

4.9 Inference

- The 4 best performing models are KNN, Random Forest, SVM and XGBoost. Out of these, KNN model produces the best results in the multi-label classification task with an accuracy of 83% on the PCA-reduced data.
- There are high rates of misclassification for some labels which have similarity in their respective hand gestures.

Chapter 5: Proposed Approach: Bucketing & Level-wise Classification

5.1 Idea & Motivation

High misclassification rates were observed for similar hand gesture labels. Bucketing "closely resembling" hand gestures to perform level-1 classification followed by classification in the combined labels as level-2 classification by models trained only on the combined label group.

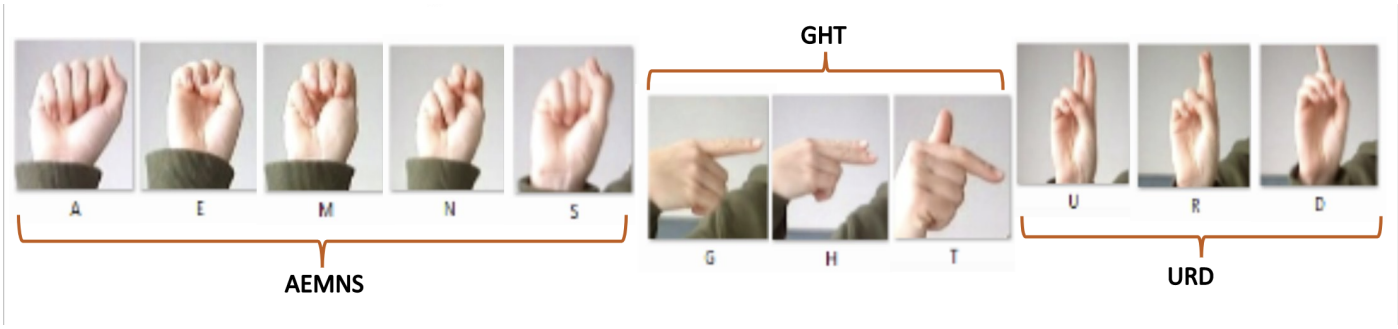

Figure 5.1: Combining Similar Hand Gesture Labels

```
"AEMNS"  "B"   "C"   "URD"  "F"   "GHT"  "I"   "K"   "L"
"O"      "P"   "Q"   "V"    "W"   "X"    "Y"
```

Figure 5.2: Combined Labels

Possible advantages:

- Idea mimics real-life decision-making intuition
- Differentiation between distinct classes followed by classification by models trained on the 'similar' labels to potentially perform better than a model trained on all labels.
- Improvement in mis-classification rate for closely-resembling labels

Possible concerns:

- Unclear demarcation of labels as "closely-resembling" vs "distinct" and hence, uncertainty around which labels to combine
- Quantifying the 'overall' accuracy of the approach to compare its efficiency with a single classifier result on all 24 labels

5.2 Procedure

- **Step-1:** Grouping Similar Hand Gesture Labels into a Combined Label, reducing total labels from 24 to 16:
- **Step-2:** Training level-wise classifier
  - Level-1 Classifier to classify into 16 labels
  - Level-2 Classifiers (3) to classify from combined labels ('AEMNS', 'GHT', 'URD') into individual labels respectively.

5.3 Pipeline

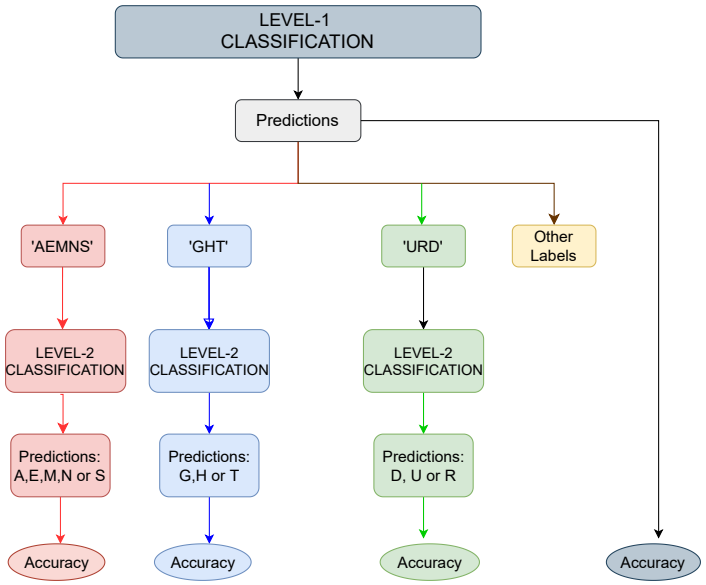

Figure 5.3: Pipeline

3 different models (KNN, Random Forest and XGBoost) were evaluated on the combined original and reduced-PCA data at every level. Best models across each level and group were selected based on the accuracy to decide the final pipeline. For original data, KNN-(RF-RF-KNN) & for PCA data KNN-(RF-KNN-RF) is the final pipeline.

- **Comparing accuracies across XGBOOST, KNN and RF on the combined original data**

ALGORITHM	LEVEL-1	LEVEL-2 (AEMNS)	LEVEL-2 (GHT)	LEVEL-2 (DRU)	OVERALL
XGBoost	83.7 %	80.0 %	89.2 %	74.9 %	71.8 %
KNN	**87.8 %**	80.8 %	91.8 %	**78.7 %**	**81.8 %**
Random Forest	82.2 %	**81.2 %**	**92.1 %**	78.7 %	77.3 %

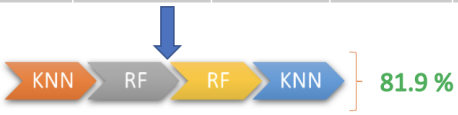

Figure 5.4: Original Data: Final Pipeline

- **Comparing accuracies across XGBOOST, KNN and RF on the combined reduced-PCA data**

ALGORITHM	LEVEL-1	LEVEL-2 (AEMNS)	LEVEL-2 (GHT)	LEVEL-2 (DRU)	OVERALL
XGBoost	83.7%	81.5 %	81.7 %	81.5 %	73.4 %
KNN	**89.3 %**	81.7 %	**93.6 %**	81.1 %	**83.8%**
Random Forest	80.1 %	**82.2 %**	88.8 %	**85.5 %**	74.2 %

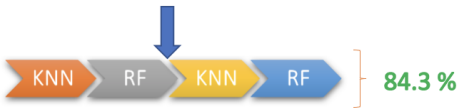

Figure 5.5: PCA-reduced Data: Final Pipeline

5.4 Results

Label	Single Classifier (KNN)	Level-wise Classifier (KNN->RF-RF-KNN)
A	0.0 %	0.0 %
E	2.2 %	7.4 %
M	43.9 %	25.1 %
N	30.9 %	45.7 %
S	19.5 %	18.2 %
G	60.3 %	6.0 %
H	2.5 %	2.5 %
T	25.4 %	13.7 %
U	27.1 %	27.0 %
R	42.3 %	0.6 %
D	7.8 %	8.9 %

- Using the proposed approach, there was on average a 7.9% reduction in mis-classification rates (in the above table) of similar hand gesture labels (8 labels out of 11).Thus, a model trained only on 'similar' labels differentiates better than a model trained on all labels.
- Accuracy was increased from 83% using KNN to 84.3% with proposed approach on PCA-reduced data. Thus, selecting best-performing models for each level of classification performs better than using a single classifier.

Bibliography

[GV16] Brandon Garcia and Sigberto Alarcon Viesca. Real-Time American Sign Language Recognition with Convolutional Neural Networks. 2016.

[MP20] Hrishikesh Patil Ashutosh Raut Prof. S S Jadhav Mohit Patil, Pranay Pathole. Indian Sign Language Recognition. *International Journal of Scientific Research Engineering Trends*, 6(4):2395–566X, 2020.

[Rat18] Dhruv Rathi. Optimization of Transfer Learning for Sign Language Recognition Targeting Mobile Platform. *International Journal on Recent and Innovation Trends in Computing and Communication*, 6(4), 2018.

[SS21] Akitaka Matsuoka Md. A.M. Hasan Shin, Jungpil and Azmain Y. Srizon. American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation. *Sensors*, 21(17):5856, 2021.