Project Title: Hand Postures (from Motion Capture)

Data Set: Hand Postures (from Motion Capture) Dataset

Sourabh Tirodkar (tirodkar@usc.edu)

USC ID- 3589406164

EE559- Mathematical Pattern Recognition

Date of submission: 8th May 2020

Table of Contents

1. Abstract

In this project, the dataset of hand postures from motion capture is present with the marker coordinate values for each posture and signifying the hand posture. The dataset has 12 markers with each one gives values in X-Y-Z coordinate system. The goal is to model the pattern recognition system and classify the postures on the test data.

The data set present is purely in the raw form which cannot be used directly for the training and testing purpose. The major task is to extract the features from this raw form, which is done by using mean, standard deviation, maximum and minimum of each X, Y, Z coordinate of the markers to form an informative 13 feature space.

Since the 13 dimensions are also a bit too many for complexity, it was reduced further using Linear Discriminant Analysis over PCA by comparing both. The reduced feature dimension was chosen wisely to 4 after trail and error and checking results on reduced dimension. Additionally, standardization was done to get the final dataset for the train and test which are now ready for building the model.

The project used 7 classifiers in total with 2 baseline classifiers of dummy classifier and Naïve Bayes. Other classifiers used were Support Vector Machine (SVM), Random Forest (RF), K-Nearest Neighbor (KNN), Stochastic Gradient and Multi-layer Perceptron Learning algorithm.

The parameters of each of the classifier was chosen with cross validation technique looping the parameters to find the best ones. The in-depth comparison of the classifiers was done by calculating the training and testing accuracy along with finding the precision (P), recall (R) and F score which helped me to make the decision that SVM gives better performance on the Posture dataset with testing accuracy of 95%. The confusion matrix was also plotted to know better about each class and finding the source of error to further investigate on.

In conclusion, the goal of modelling the hand posture dataset for prediction of hand postures was done.

2. Introduction

2.1 Problem Statement and Goals

There are 5 types of hand postures from 12 users which were recorded using unlabeled markers on fingers present on the glove. The markers are unlabeled so, which marker position gives what values are unknown. Each marker gives X, Y and Z coordinate values. The goal is to consider the user dependent posture recognition problem and classify the posture for the users that are not trained on. There are 12 markers in total meaning the total data points noted could be till 36. Not all the markers have the values for one posture position meaning the features can be less than 36 for 1 position as well. The minimum features are 9 for one position.

Below, is the construction of hand gloves arrangement and the marker positions.

Fig. 1 Glove model used for noting values

In a nutshell,
Total users = 12
Total Classes = 5
Training Data Users = 9
Testing Data Users = 3
Each training user has 1500 data points, N_train = 1500 * 9 = 13500
N_test = 21099
Max Features = 12 (No of markers) * 3 (Each marker-X, Y and Z) = 36

The final goal of the project is developing a pattern recognition system that will operate on the real-world dataset and will be used for prediction, meanwhile applying the techniques and tools learned.

3. Approach and Implementation

The following section has in-detailed explanation beginning from the feature extraction, reduction to final testing on different classifiers and reasoning for everything chosen.

3.1. Feature Extraction

One of the main challenges of the Hand Postures dataset is the feature extraction from the features provided. The data is in the raw form and thus wouldn't work for recognition problem. The features are randomly between 9 and 36. It was important to define new features and then work on those.

I decided to stick to the suggested 13 features extracting from the dataset. The first feature selected was the number of recorded markers. Not all users have all the data points filled. Thus, it was important to take this feature into consideration. The value

of this feature would be always be in a multiple of 3 and between 9 and 36. For example- if a specific posture has 5 markers noted, meaning 15 data points values are present, each x, y and z coordinates.

After this, I combined all the X, Y and Z markers to extra more features from them. I used mean, standard deviation (SD), maximum and minimum as the rest of the features.

The data also had NAN values which had to be considered for all those above features. The mean, standard deviation, maximum and minimum was done for each combined X, Y and Z coordinate forming remaining 12 features.

Thus, my final 13 features extracted were:

1- No of Recorded markers
2- Mean of combined X
3- Mean of combined Y
4- Mean of combined Z
5- SD of combined X
6- SD of combined Y
7- SD of combined Z
8- Maximum of combined X
9- Maximum of combined Y
10- Maximum of combined Z
11- Minimum of combined X
12- Minimum of combined Y
13- Minimum of combined Z

Special care was taking not considering NAN values as say for 7 markers values, it would be unfair to divide by 12 rather than 7 for the mean disturbing the features. Similarly, for the values with negative values, the minimum and maximum was considered accordingly.

3.2. Pre- Processing

Pre-Processing of data is important as the noted values may be out of range. It does have the direct impact on the success rate of the model. The complexity of the model also reduces significantly. I have used sklearn preprocessing technique for the standardization of the data. It uses mean and standard deviation for each of the data.

The mean of the modified data is 0 while the standard deviation is 1. This is very significant in handling the data for the model training. The test data has been standardized using the mean and variance of the train data itself as test data isn't available in the real world.

I also tried the comparison between the non-standardized data and standardized data which are as follows:

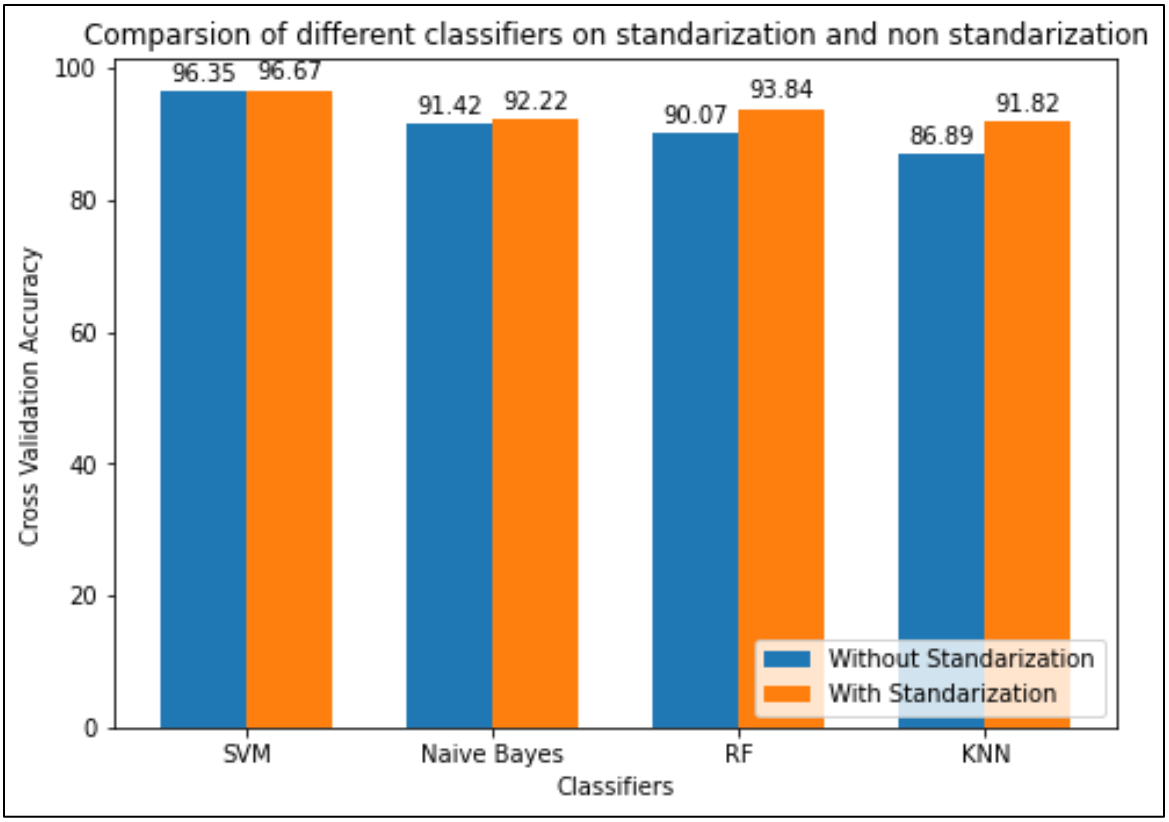


Fig. 2 Comparison of standardization and non-standardization.

I used 4 classifiers for the comparison and plotted the test accuracy for the standardized data and unstandardized data. As seen, the test accuracy for standardized data is more in every case and thus explains the reason why pre- processing is necessary.

3.3. Feature dimensionality adjustment

Feature dimensionality adjustment whether reducing or expanding the features plays a huge role in recognition systems. I have used feature reduction a.k.a. dimensionality reduction.

It is the process where the total number of features are reduced in a heavy computation environment without losing important feature information. This makes the system's work easier and faster. Before feature reduction, my code was taking about 10 mins to run for all the classifiers (13 features), after reducing it to lower dimension, it is now running in 5-6 mins. Thus, lot of computation time is saved.

There are various methods to do so like Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), etc. I have used LDA for the same. LDA is also known as Fisher Discriminant Analysis (FDA). The reason I chose LDA over PCA as PCA is an unsupervised dimensional reduction technique unlike LDA which is a supervised one. PCA also works on smaller dataset, and as the dataset had different class separability

thus, I used LDA. I assumed the cross-validation accuracy of PCA would be lower than LDA and which was seen exactly during my implementation.

Below is the comparison of LDA and PCA for dimension 4 for some of the classifiers where cross validation accuracy is as shown:

Classifier	**LDA**	**PCA**
SVM	96.67%	72.01%
Naïve Bayes	92.22%	67.41%
RF	93.77%	68.36%

LDA

```
1. SVM Classifier
Max Cross Validation Accuracy: 0.9667407407407408
2. Naive Bayes Classifier
Max Cross Validation Accuracy: 0.9222962962962963
3. Random Forest Classifier
Max Cross Validation Accuracy: 0.9377037037037037
```

PCA

```
1. SVM Classifier
Max Cross Validation Accuracy: 0.7201481481481481
2. Naive Bayes Classifier
Max Cross Validation Accuracy: 0.6741481481481482
3. Random Forest Classifier
Max Cross Validation Accuracy: 0.6836296296296296
```

LDA tries to preserve as much as discriminatory information from the data. It tries to find the directions along which the classes are best separated by taking into consideration the scatter of within- class and of scatter between-classes.

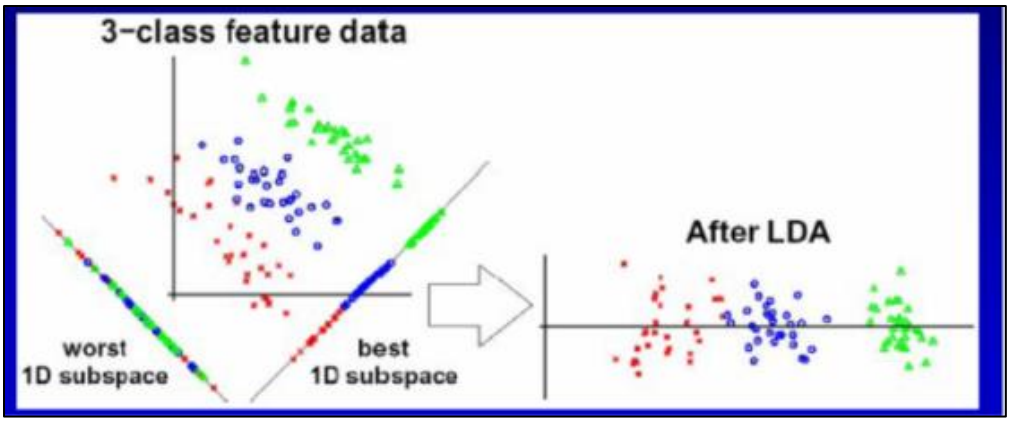


Fig. 3 Working of LDA

Algorithm:

$$J(w) = \frac{|m1 - m2|^2}{S1^2 + S2^2}$$

This is the criterion function that we get, and we must maximize this w.r.t *w.* After solving we get FLD solution as: $w = Sw^{-1} * (m1 - m2)$

Where m1, m2 are the mean for 2 class problem. FDA can be expanded to muti-class problem by following the below algorithm.

To convert D dim to d dim:

1. Applying FLD for D dimension to 1 D, result w(1)
2. Loop k= 1 to (d-1) and subtract wT*x from each data point.

3. Working in subspace uk which is orthogonal to w(k).
4. Finding FLD for (D-k) dim to 1D to find result of w(k+1). The (k+1) becomes the new feature space.

Choosing the right dimensionality:

I used sklearn inbuilt function for LDA where I was supposed to mention the number of reduced feature dimension. I choose 4 as the reduced feature space. The documentation for LDA gives n_components = min (n_features, n_class-1) as the optimal value. In our case, it is min (13, 4) = 4.

I also tried to reduce it to 3 dimensions, but the result was not that good. So, I sticked to choosing 4 as reduced feature space and decided to work ahead with this.

3.4. Dataset Usage

The procedure that I followed for the Posture dataset are in the following order:

3.4.1. Feature Engineering

Posture dataset had 9 to 36 features for each of the data point. I extracted the feature as described in section [3.1] to make the number of features to 13. The training data has 13500 samples for train while 21099 samples for test. So, my final matrix after feature extraction: **For train - 13500*13, For test – 21099*13**

3.4.2. Pre-Processing:

Pre-Processing was necessary as without normalization and standardization, the results were not good as described in section [3.2]. I followed standardization on the train data as well on the test data using training samples only. The output for the same had mean of 0 and standard deviation of 1 for each of the columns. This step helped me reduce my running time and made sure the output isn't out of range for any value. The dimensions remained same from the feature extracted output.

3.4.3. Dimensionality Reduction:

As discussed in section [3.3], I used LDA for feature reduction. The major part for choosing the best dimension to reduce for. I tried using 3 values (dim- 3,4 and 7). As the dimension were increasing the runtime was also increasing. I found the optimal runtime and accuracy on the reduced dimension of 4. I used Singular Value Decomposition (SVD) in LDA to extract the most prominent features from all the 13 features. The output after dimensionality reduction:

For train data: - 13500 * 4, For test data: - 21099 * 4.

3.4.4. Model Training, Selection and Cross-Validation set:

So, once the final data set are ready, we are ready to start training the model. The major step was Cross-Validation step. As suggested, I used *'LeaveOneGroupOut'* for splitting the data.

Here it is how I worked out. I knew from the raw data that training data has 9 users which was used for the splitting the data. Each user in the training set had 1500 samples, so we had 9 groups each with size 1500 samples.

Using sklearn, '*LeaveOneGroupOut*' for splitting I made sure to remove one user out which was used for cross validation and rest of the remaining data for training purpose. Meaning, entire 1500 samples from 1 user were used for cross validation and rest 12000 samples (13500-1500) were used for training.

As the number of groups were 9, there were 9 folds used for the cross validation. Each cross-validation set had dimension of 1500*4 while each training data had dimension of 12000*4. The cross-validation was used to choose among the best hyperparameter for the classifier. The runs of the cross-validation dataset were based on the range that I am choosing for each parameter.

Each classifier has 1-2 parameters in it which can be tuned to get the best results. I used one parameter for each classifier for this. This parameter was varied in a specific range, let say C value in SVM in range of 10^-2 to 10^2. I selected the 20 random points in this range, passed this value into training and cross-validation function. Here, 9-fold were present and 20 runs as I picked 20 C values.

As each value say C value of 1 would go into the function, run the code 9 times, each time different group for validation set. Then, I found out the average of all 9 values and stored into an array. Thus, if my range of C takes 20 values, I would have 20*1 matrix each row having average cross validation accuracy for each C.

The decision on cross validation accuracy was to choose the maximum accuracy from the same matrix and then found out for what parameters of the classifier does it achieve. This is important since I now have the best hyperparameter value used in each of the classifiers. None in the process, the test data was ever used.

Now, we are all set to use the best value in each of the classifier chosen. I choose the classifier, already knowing the best parameter, setting this value into classifier again, training on originally train data which had dimensions of 13500*4. Note, I am not splitting the data henceforth. The model is trained on the entire train data and then checked onto the test data which is of dimensions 21099*4. This is the first time I am using the test data and it is only used once which is at the last step. The accuracy obtained on the test data is the final accuracy that is most optimal in that classifier among the range of parameters in the classifier.

Exact same process is followed for all the number of classifiers chosen and then testing to get the optimal test accuracy.

3.5. Training and Classification

The procedure for training the model was chosen as mentioned in section [3.4.4]. I used 7 classifiers in total including the random classifier which was told to use for the baseline model. The classifiers that I used are as follows:

1. Random (Dummy Classifier)
2. Support Vector Machines (SVM)
3. Naïve Bayes Classifier
4. Random Forest (RF) Classifier
5. K- Nearest Neighbor (KNN) classifier
6. Stochastic Gradient descent
7. Multi-layer Perceptron (MLP) Classifier

For each of the model, the classifier parameters were chosen from the cross-validation method as described in the above section.

3.5.1. Random Classifier

A random classifier is a classifier that outputs class labels without looking at the inputs. It is a classifier which is often used for the simple baseline and to compare the other classifiers. I have used sklearn dummy classifier for the implementation of the same. I have used strategy as 'most frequent' to predict the most frequent label. As it is a baseline system, the train and test data were directly fed into the model without cross validation. As the model is only for baseline, the performance is not so good. Below is the result of the dummy classifier.

```
7. Dummy Classifier
Accuracy train RANDOM: 0.2228888888888889
Accuracy test RANDOM: 0.1676856723067444
.........
```

Train Accuracy - 22.28%, Test Accuracy – 16.76%

3.5.2. Support Vector Machines (SVM)

SVM is a supervised learning model that is widely used for classification problem. It can classify the linear as well as the non-linear data. The main working of SVM is finding the best separating hyperplane with the latest margin and to maximize it.

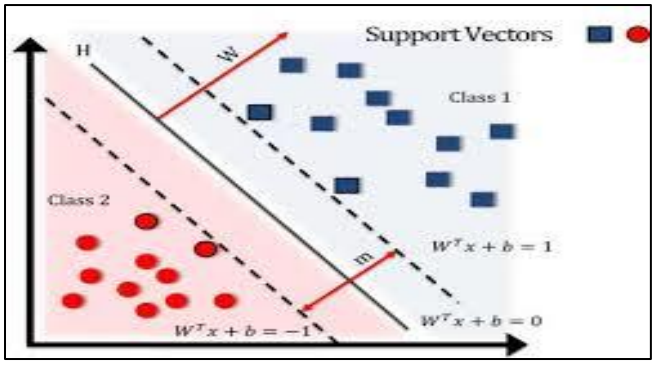


Fig. 4 SVM Algorithm

Algorithm: zi$*$ $(wT * ui + w0) \geq 1 \; \forall i \;\; with \; minimal \; ||w||$
It also uses Lagrange Optimization where the task is to minimize criterion function with respect to above constraint.

For best results, it is necessary to tune the parameters in sklearn SVM function. I have used RBF kernel as it was giving me better results than linear kernel. The parameters are C and gamma. I have set gamma to 'auto' while changing the C in range and passing through the cross validation as discussed in-detailed in the previous section. Here, C is a penalty for misclassification which when C is small, the misclassified points won't make a huge impact, but when C is large the classifier penalizes the misclassified data point and tries to avoid it. The C value which gives better result is 0.215 which is chosen from the range of 0.001 to 100.

```
1. SVM Classifier
Max Cross Validation Accuracy: 0.9667407407407408
C value 0.21544346900318834
Accuracy train SVM: 0.9982222222222222
Accuracy test SVM: 0.9500450258306081
```

Training Accuracy: 99.82%, Testing Accuracy: 95.004%

Below is the confusion matrix for the SVM classifier:

```
[[4203   48  145   68    2]
 [  26 4253   50   40   33]
 [ 123    0 4513  143    0]
 [   0    0  118 3758   38]
 [   0   72   56   92 3318]]
```

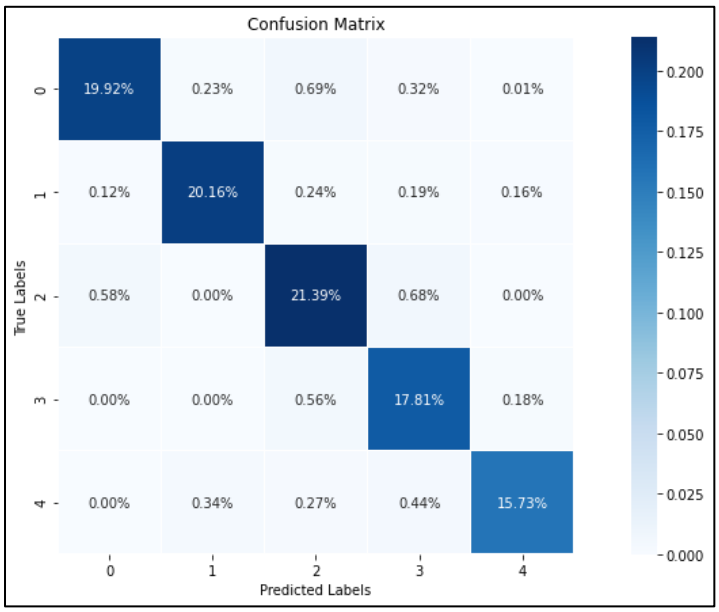


Fig. 5 confusion matrix for SVM

3.5.3. Naïve Bayes Classifier:

Naïve Bayes Classifier is a probabilistic classifier which is commonly used classifier in Machine Learning which uses Maximum A Posteriori decision rule in a Bayesian network. Here, as told, Naïve Bayes is used as a baseline model.

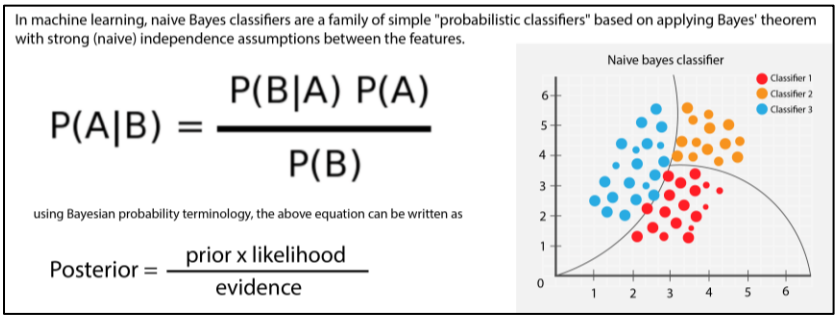


Fig. 6 Bayes Classifier

I have used sklearn function for implementation (GaussianNB) which has the parameter of variance which I has range from [10^-4, 10^2] in randomly chosen 20 points between those. The best variance is chosen from the max cross validation accuracy which is 0.01. Using this value of variance, I retrained the model on training data and then tested to get the following accuracy.

```
2. Naive Bayes Classifier
Max Cross Validation Accuracy: 0.9222962962962963
Best Variance value 0.01
Accuracy train Bayes: 0.9558518518518518
Accuracy test Bayes: 0.8695198824588843
```

Training Accuracy: 95.58%, Testing Accuracy: 86.95%

The confusion matrix for the same is as below:

```
[[ 4116   48  294    0    8]
 [    0 4199   64   54   85]
 [    0    0 3890  889    0]
 [    0    0 1112 2802    0]
 [    0   10   78  111 3339]]
```

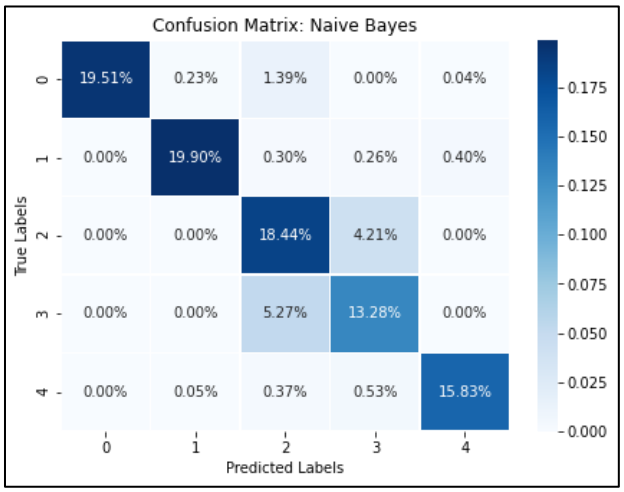


Fig. 7 Confusion matrix for Naïve Bayes

3.5.4. Random Forest (RF):

RF is a type of supervised learning algorithm which uses decision trees for the classification. It fits several decision trees on various sub samples of data and uses averaging to improve the predictive analysis.

Random Forest is considered as one of the most accurate classifiers among others, handles large data but has a disadvantage of overfitting and is often time consuming.

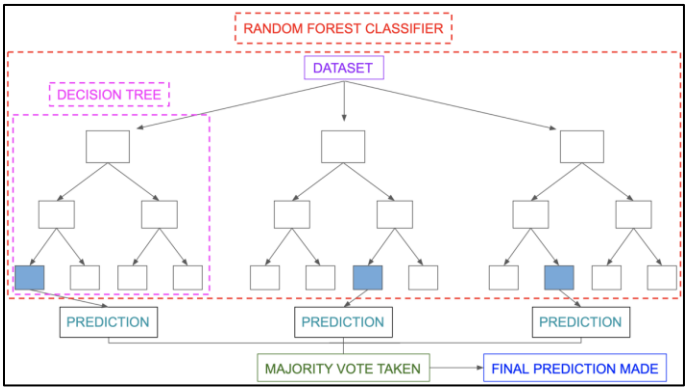


Fig. 8 Random Forest decision trees

I have sklearn function for RF which is 'RandomForestClassifier' in which the n_estimators value is varied in cross validation split in a range from 1 to 100 taking 20 random samples from it. The output of RF changes for every run and I tried to get the latest result which is as below:

```
3. Random Forest Classifier
Max Cross Validation Accuracy: 0.9377037037037037
Best estimator value 73
Accuracy train RF: 1.0
Accuracy test RF: 0.8310346461917626
```

Training Accuracy: 100%, Testing Accuracy: 83.10%

There is high chance that my training accuracy has been overfit, I did try to tune more hyperparameters but couldn't spend much time on that.

The confusion matrix for RF is as below:

```
[[4383   48   20   15    0]
 [  25 3846    1   31  499]
 [   6  292 3979  477   25]
 [   0  540 1228 1914  232]
 [   6    8   47   65 3412]]
```

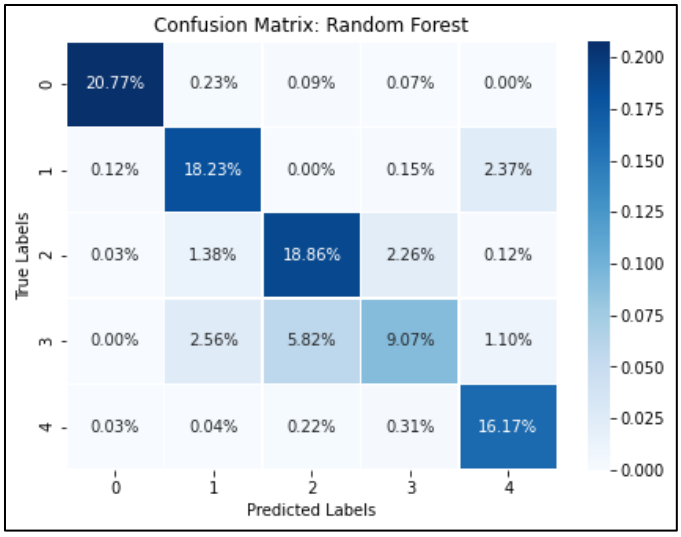


Fig. 9 Confusion matrix for RF

3.5.5. K-Nearest Neighbor (KNN)

KNN is a type of supervised statistical classifier where the data consists of k closest training samples in the feature space. It classifies the data using the k- nearest neighbor vote. The main challenge is to choose the k value to form the clusters.

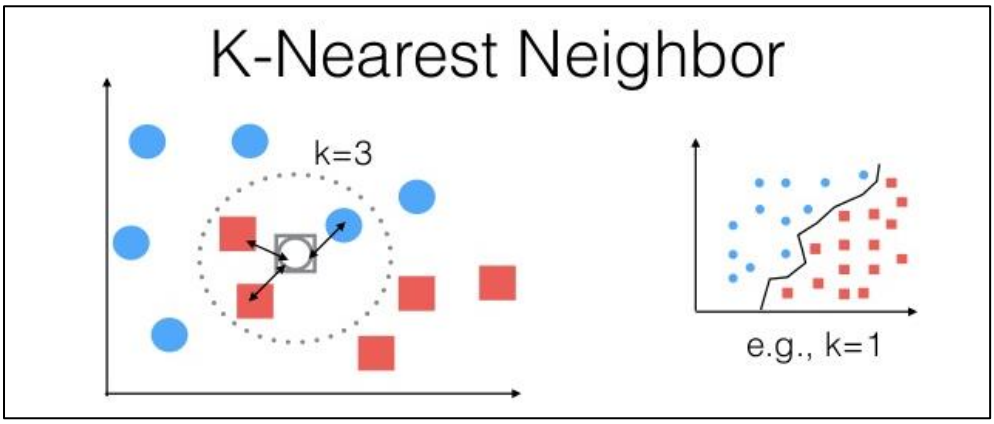


Fig. 10 KNN algorithm describing clusters

In sklearn, 'KNeighborsClassifier' function has been used which has the obvious parameter of k- neighbor to specify. As in all the classifiers, I varied the k value from 1 to 100 with 20 randomly chosen points. The training and cross validation was done using different values and max accuracy was noted to trace back the best k-neighbor value. For my implementation, I got the value of k as 6. Using best neighbor value, I retrained the model on train data and then transformed into test data to get following accuracy.

```
4. KNN Classifier
Max Cross Validation Accuracy: 0.9182222222222222
Best estimator value 6
Accuracy train KNN: 0.9971111111111111
Accuracy test KNN: 0.8489027916014977
```

Training accuracy: 99.711%, Testing Accuracy: 84.89%

Also, I have plotted the confusion matrix for the same.

```
[[4211   48  201    6    0]
 [  28 3525   61   31  757]
 [ 153    2 4448  175    1]
 [   0   47  452 2398 1017]
 [   0  124    2   83 3329]]
```

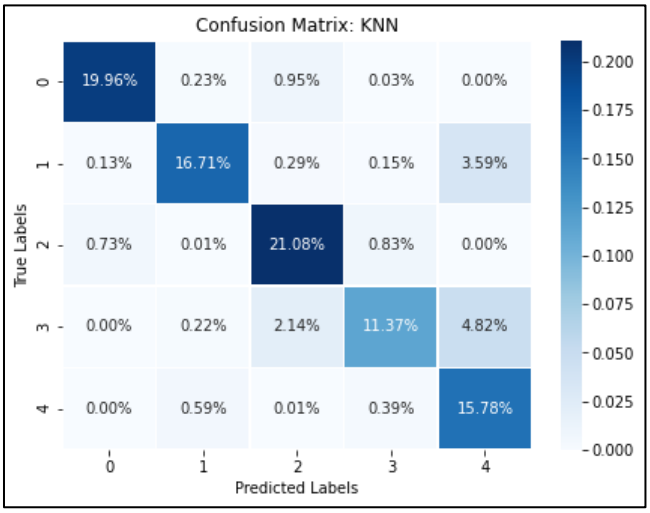


Fig. 11 Confusion matrix for KNN

3.5.6. Stochastic Gradient Descent (SGD):

SGD is a simple discriminative linear classifier which uses gradient descent optimization for the learning. It is mostly used for the linear data; I used the same as I wanted to try how linear classifier would work on the posture dataset. The gradient loss is estimated by mini batch method.

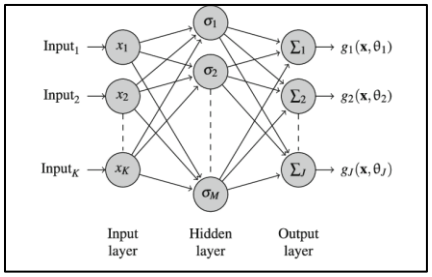
The training and testing accuracy for the data are as below:

```
5. SGD
Accuracy train SGD: 0.9425925925925925
Accuracy test SGD: 0.9215602635195981
```

Training Accuracy: 94.25%, Testing Accuracy: 92.15%

3.5.7. Multi-layer Perceptron (MLP) Classifier:

The MLP architecture is as shown below. It has the input layers, the hidden layers and the output layers equal to the number of classes. It also used the activation function, which is chosen 'relu' in most case. [3]



I used Sklearn MLP function for the implementation and tuned the parameters to hidden layers as 100, with Adam optimizer, activation function relu and learning rate to be constant. There are lot of parameters that can be tuned, and higher accuracy can be achieved. The results of my implementation of MLP is the following:

```
6. MLP
Accuracy train MLP: 0.9980740740740741
Accuracy test MLP: 0.9102801080619934
```

Training Accuracy: 99.80%, Testing Accuracy: 91.02%

4. Analysis: Comparison of Results, Interpretation

Among the classifiers that I used; my best results were found in Support Vector Means (SVM) as seen from the table below which shows the accuracy for all the classifiers.

	Average Cross Validation Accuracy	Training Accuracy	Testing Accuracy
1. SVM	96.67	99.82	95.04
2. Naïve Bayes	92.22	95.58	86.95
3. Random Forest	93.77	100	83.10
4. KNN	94.27	99.71	84.89
5. SGD	91.61	94.25	92.15
6. MLP	88.49	99.82	89.26

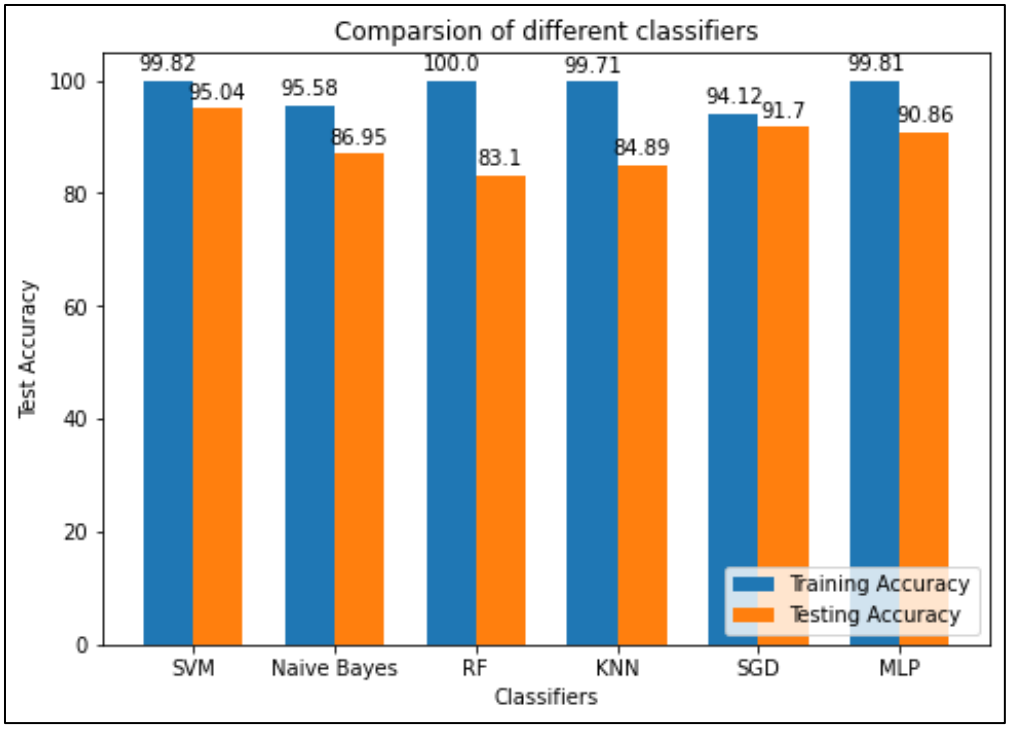


Fig. 13 Comparison of different classifiers on train and test accuracy.

Thus, SVM is chosen as the final model for this Hand Postures dataset.

Below is the precision, recall, F score for some of the classifiers.

SVM

	precision	recall	f1-score	support
1.0	0.96576	0.94111	0.95328	4466
2.0	0.97256	0.96615	0.96934	4402
3.0	0.92442	0.94434	0.93427	4779
4.0	0.91636	0.96014	0.93774	3914
5.0	0.97847	0.93782	0.95771	3538
accuracy			0.95005	21099
macro avg	0.95151	0.94991	0.95047	21099
weighted avg	0.95078	0.95005	0.95019	21099

Naives Bayes

	precision	recall	f1-score	support
1.0	1.00000	0.92163	0.95922	4466
2.0	0.98638	0.95388	0.96986	4402
3.0	0.71534	0.81398	0.76148	4779
4.0	0.72666	0.71589	0.72124	3914
5.0	0.97290	0.94375	0.95811	3538
accuracy			0.86952	21099
macro avg	0.88025	0.86983	0.87398	21099
weighted avg	0.87743	0.86952	0.87231	21099

Random Forest

	precision	recall	f1-score	support
1.0	0.99339	0.97559	0.98441	4466
2.0	0.70488	0.87029	0.77890	4402
3.0	0.74317	0.79682	0.76906	4779
4.0	0.60941	0.31451	0.41490	3914
5.0	0.82075	0.95902	0.88452	3538
accuracy			0.78772	21099
macro avg	0.77432	0.78325	0.76636	21099
weighted avg	0.77634	0.78772	0.77036	21099

KNN

	precision	recall	f1-score	support
1.0	0.95879	0.94290	0.95078	4466
2.0	0.94100	0.80077	0.86524	4402
3.0	0.86135	0.93074	0.89470	4779
4.0	0.89046	0.61267	0.72590	3914
5.0	0.65223	0.94093	0.77042	3538
accuracy			0.84890	21099
macro avg	0.86077	0.84560	0.84141	21099
weighted avg	0.86893	0.84890	0.84827	21099

The tabular form for the same is given below and we can verify by looking at Precision (P), Recall (R) and F score (F), that SVM is better.

	SVM			Naïve Bayes			RF			KNN		
	P	R	F	P	R	F	P	R	F	P	R	F
Label 1	0.96	0.94	0.95	1	0.92	0.95	0.99	0.97	0.98	0.95	0.94	0.95
Label 2	0.97	0.96	0.97	0.98	0.95	0.96	0.70	0.87	0.77	0.94	0.80	0.86
Label 3	0.92	0.94	0.93	0.71	0.81	0.76	0.74	0.79	0.76	0.86	0.93	0.89
Label 4	0.91	0.96	0.93	0.72	0.71	0.72	0.60	0.31	0.41	0.89	0.61	0.72
Label 5	0.97	0.93	0.95	0.97	0.94	0.95	0.82	0.95	0.88	0.65	0.94	0.77

We can also see that there is value difference for P, R and F for labels 4 and 5. This is mostly because we have smaller amount of data present on those samples. This is known as imbalance number of classes. This can be improved when there is balance among all the labels, but most likely the accuracy would drop down.

<u>Comparison with baseline system:</u>

1. Dummy Classifier:
   The train and test accuracy results are given in the previous section and comparing with the above table, the rest of the classifiers results are greater than dummy classifier.

2. Naïve Bayes:
   Naïve Bayes was told to use as a baseline system. It is seen that most of the classifiers test accuracy are greater than the Naïve Bayes classifier. Also, Naïve Bayes used here is tuned to get best value using cross validation.
   From what I observe, was most of the classifiers had good performance considering the feature extraction, standardization and reduction done before. I found that Random Forest Classifier was changing every time I run the code. After digging into the problem, I found out that as there as various trees in RF, and each tree is only trained among randomly selected samples from the train set which might be because of repetition and choosing random subset of features.

Also, I was surprised by the result of SGD classifier because being a linear classifier it performed well on the training and test data. Initially, I thought being linear the accuracy would be low, but gathering information on the same cleared my doubt that uses stochastic gradient descent for the optimization which is done using mini batch and thus gives better results.

5. Summary and Conclusions

In this project, I learned a lot of things and mostly I am satisfied as the main goal for the project which was set was achieved to make a pattern recognition system for the classification of real-world dataset, here the hand posture dataset from motion capture.

The key findings that I take out from this project was the importance of feature extraction and how to deal with the raw data and reasoning why the same cannot be directly used for the training purpose. Additionally, the feature extracted data have a lot to do for the classifier to run it without any complexity meaning the significance of standardization and feature dimensionality reduction.

The in-depth analysis of every classifier along with usage of cross validation helped me chose best parameters and achieve the higher accuracy.

For the follow-on work, as I tried to deal with the problem of user independent data, some modifications in the algorithm or more information regarding user's identification would be done to predict the user too making it user dependent classification problem.

Moreover, the theoretical knowledge, which was taught in EE559 regarding classifiers, dimensionality reduction, etc., I was able to implement it on the real-world data which almost most of the problems are once I step up in the industry.

References:

1. https://www.reddit.com/r/learnmachinelearning/comments/8eztes/why_is_it_called_a_random_forest/
2. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
3. https://en.wikipedia.org/wiki/Multilayer_perceptron
4. https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d
5. https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf