Human Activity Classification
Team-13

G Vinay Chandra (2020101010)

- *SVM with reduced features and full features*

Y Surya Teja Reddy (2020101042)

- *Decision Trees(boosting+bagging+single)*
- *Preprocessing*

P Neeraj (202010126)

- *Logistic Regresssion for multi-class classification*

T Sai Venu Gopal (2020101067)

- *Deep learning*
- *Preprocessing*

Data Cleaning

We used PAMAP2 dataset from the UCI repository of machine learning datasets. This dataset would contain raw 9-axis IMU datastreams (from hand, chest and ankle) and also ahs heartrate. We have 1.9 million data points each containing 52 features. We ignored the orientational features which reduced the number of features to 40. The first column was also excluded as it represents the time stamp. We had to extrapolate the heart rate values. This is because of the frequency differences in the sensors used to measure different attributes. We extrapolated the heart rate linearly. So all the NAN values between two different heart beats would be linearly assigned to values. This is a good approximation because there would be slight change in heart beat within a matter of milli seconds. We also had other features containing NAN values. We have removed these datapoints and used the rest of the datapoints as our dataset. We have divided oru dataset in 80% training data and 20% testing data. We also removed the datapoints whose label was given to be 0. This was done in the research paper too. We removed 12 orientation features which were, orientation = [19,18,17,16,36,35,34,33,53,52,51,50]. We also had to get features corresponding to only hand, chest, ankle (each of which had a size of 14).

A separate file was used to do data cleaning which was then used in individual models to get data. This file returned the Data_NT, Data_NO_NT_Hand,Data_No_NT_chest, Data_No_NT_ankle. We normalized the values by subtracting each value by its mean and dividing it by the variance. This was done in SVM to increase accuracy. Neural networks didn't need normalisation; any float value is accepted by neural networks. Accuracy didn't change for the neural network even after normalizing (tried with a small dataset). We have done the 5-fold validation for bagging and boosting. K fold gives a set of scores (here 5) whose variance and mean are calculated and the model is selected to be the one with lease 1se i.e mean plus one standard deviation.

<u>Deep Learning</u>

We have implemented multilayer perceptron for the multi-class classification. A multilayer perceptron has many hidden layers and hidden nodes in each layer. It initially takes in ‘n’ features as inputs and outputs ‘k’ sized array which indicates the probability of the given sample to belong to classes. We have about 40 features as input and about 25 classes (Actually 12 but to have continuity we used 25 outputs). So our standard neural network contains 3 hidden layers each having 100 nodes (We changed these and checked accuracies for other combinations too). We also have a dataloader which would batch our dataset into batches of 100 size. This way we applied gradient descent for every batch. So for our 1.9 M datasets, (80% of which is training) 19k gradient descent operations. This would help the loss converge instead of considering the whole dataset and applying back propagation. The standard activation energy used is ReLU between layers (we changed this and checked the results too). We also used the dropout with probability 0.5 which helps in the overfitting problem.The loss function we used is CrossEntropyloss. This is the best loss for multi-class classification. The loss of this is given by

$$H(p,q) = -\sum_{x\in\text{classes}} \overset{\text{True probability distribution (one-hot)}}{p(x)} \log \underset{\text{Your model's predicted probability distribution}}{q(x)}$$

This actually distinguishes the outputs better. For example, if the outputs were [0.98,0.01,0.01] and [0.51,0.48,0.01] and the class label was 0, clearly the first one has less loss compared to the second output. This will be taken care by the cross entropy loss. In our dataset we removed the timestamp and also the orientations(was advised in the paper to do so). The final dataset contains label at 0th index and the rest values are features (inputs). We have to consider one-hot encodings of the labels which is a vector with zero values everywhere except at the index of the class. I.e For class 1, this vector would have value zeros every except at index 1. This way the loss would just become the negative log of the probability of that class. Also the probabilities must add up to 1 so softmax layer is used (In our code the loss function takes care of it). Now that we have the loss we use optimiser which does backpropagation. We ran this backpropagation many times with varying number of epochs.(Results of these are shown too). After all this we found out the test data accuracies by comparing the output from the model for batched test input. Batched input reduces burden on the CPU at a given time which is more efficient. Our research paper also did an analysis by dividing the features based on the body part from which the readings originated from like hand, chest and ankle. We would have reduced number of features for these (approx 15 for each). We can use the same code since the code is modular and just required changes in 2-3 lines. Our paper also focused on the training time. This was optimised as we took softmax before finding the loss. This helps in calculating the loss of the function easier. The optimiser we used is SDG optimiser this seemed better compared to other optimisers like Adam,Adamw.

Test result-1 :

This shows the accuracy changes after each epoch of the training set (For 3 hidden layers and 25 nodes in each layer)

Epoch Number	Test Accuracy
0	9.2%
1	87.8%
2	89.1%
3	88.4%
4	91.2%
5	92.0%
6	93.4%
7	93.9%
8	94.7%
9	95.8%
10	96.8%

Clearly as number of epochs increases, the test accuracy increases. This is because the model tunes the model parameters so that the final loss is decreased. The test accuracies sometimes decreased. This is because the loss only takes care of the train dataset. The training accuracies continuously increase but the testing accuracies might go up and down. Also sometimes the gradient descent would should out of the minimum (because of high learning rate). The model then readjusts in the next step which makes the accuracy better again. The first epoch (epoch 0) gave an accuray of 9%. This is when no training is done. There are approximately 12 classes, so the model initially guessed randomly and got 1/12 correct which can be seen by the accuracies. Then it increased drastically to 87.8%. It increased from there on.

Test result-2;

We have fixed the number of nodes (25 nodes) in a layer and the number of epochs used (10 epochs) to get the test accuracies by varying number of layers used

Number of layers	Test Accuracies	Time taken to train and test
2	92.7%	279 seconds
3	92.6%	315 seconds
4	89.2%	345 seconds
5	88.7%	376 seconds

The test accuracies have decreased on increasing the number of hidden layers. This could be because of overfitting. Also the time taken to run is increases as the number of hidden layers/hidden nodes increased so it required more calculations to get the output.

Test result-3;

We have fixed the number of layers (3 layers) and number of epochs used (10 epochs) by varying number of nodes used in a layer

Number of Nodes in a layer	Test Accuracies	Time taken to train and test
25	92.6%	315 seconds
50	96.8%	347 seconds
100	98.2%	370 seconds

We can clearly see that the number of layers mattered while getting the test accuracies. More number of layers indicates better accuracy. This is because the original function that maps the input to output had many parameters (because there are many features). To catch all of the attributes we need more nodes (parameters). Also the time taken to train and test increased as the complexity of the neural network increased.

<u>Test result-4;</u>

We have fixed number of epochs, number of hidden states and number of nodes in a hidden state and changed the activation energy between layers. (3 hidden states and 25 hidden nodes)

Activation Function	Test Accuracy
ReLU	92.6%
LeakyReLU	93.2%
Sigmoid	90.6%
Tanh	91%

There is a slight difference between the test accuracies obtained for each activation function. The ReLU and LeakyReLU performed better. This is because the final activation function. We used softmax in the end as the activation function. In sigmoid, it gives values from 0 to 1 which is similar to softmax. Similarly, tanh also gives values from 0 to 1. So in these cases the softmax is applied on a lower range making the probabilities less feasible. ReLU can have values from 0 to infinity, so the higher values (much greater than one) have higher probabilities which overshadow the other values'.

<u>Test result-5;</u>

We now check the accuracies for just features corresponding to hand, chest and ankle. We only ran this once with 3 hidden states and 25 hidden nodes. Note that the number of features also decrease. We have three neural networks each corresponding to one of the body parts' features. We also had only 15 epochs.

Body Parts	Test Accuracy
Hand	78%
Chest	72%
Ankle	70%

Highest accuracy is for hand which is 78%

Single Tree(Gini Impurity)

Single tree on the entire Dataset was of depth 73 and accuracy 99.97%. Here the hyperparameter we want to tune is height to reduce overfitting . So values of accuracy for given height are:

Depth	5	10	15	20	25	30	35
Accuracy	64.1	89.9	99.49	99.49	99.65	99.73	99.79

We see that depth of 15 and 20 have the same test accuracy, which means after that it is trying to overfit to data. Since data is from only 9 subjects , generalization is lost after a depth hence increase from 25 onwards. Hence the best depth to combat overfitting is 15 .

When we run on the limited feature set we get the best accuracy on hand which is 95.9%

Boosting(Gini,Adaboost)

For total featureset

The Basic idea is that each model should be complementary to each other. We calculate weights of each sample, update them on missclassification, give weight to each model based on its validation score and get a composite model at end that is ensemble of weak classifiers.

- Calculate the error ϵ of the training set summing over all datapoints x_n in the training set with:

$$\epsilon_t = \frac{\sum_{n=1}^{N} w_n^{(t)} * I(y_n \neq h_t(x_n))}{\sum_{n=1}^{N} w_n^{(t)}}$$

where $I(cond)$ returns 1 if $I(cond)$ == True and 0 otherwise

- Compute α with:

$$\alpha_t = \log(\frac{1-\epsilon_t}{\epsilon_t})$$

- Update the weights for the N training instances in the next $(t+1)$ model with:

$$w_n^{(t+1)} = w_n^{(t)} * exp(\alpha_t * I(y_n \neq h_t(x_n)))$$

- After the T iterations, calculate the final output with:

$$f(x) = sign(\sum_{t}^{T} \alpha_t * h_t(x))$$

Boosting total feature set (limited data points since boosting needs to be done sequentially and number of trees and depth need to be varied, which will take days of training)

We fixed number of trees 50 and changed depth parameter and their corresponding 1se values:

Depth	5	6	7	8	9	10
1se Values	90.1	91.1	93.7	94.9	95.1	95.9

There is an increase in accuracy with depth, and we only considered 10 because it should be weak and single tree itself gives very good accuracy of single tree with depth 15.

We fixed depth to be 5 and changed number of trees and their corresponding 1se values:

Num Trees	50	100	250	500
1se value	90.1	92.2	95.4	96.0

We see that the rise in accuracy is very good at last points, which means the model hasn't started overfitting, hence the optimal hyperparameters taken from the considered set of depths and number of trees is 500 trees with depth 10.
The final accuracy on total dataset comes out to be : 98.3%

For limited Feature Set(hand)

We fixed number of trees 50 and changed depth parameter and their corresponding 1se values:

Depth	5	6	7	8	9	10
1se Values	89.5	92.1	93.5	94.6	96.3	96.0

There is an increase in accuracy with depth, and we only considered 10 because it should be weak and single tree itself gives very good accuracy of single tree with depth 15.

We fixed depth to be 5 and changed number of trees and their corresponding 1se values:

Num Trees	50	100	250	500

1se value	89.5	92.6	96.8	94.8

We see a slight decrease from depth 9 to 10 and 250 to 500 which is a testament to overfitting. Hence the best model is 250 trees with max depth as 9.

For hand the final accuracy comes out to be : 93.2%

Random Forest

For total featureset

In bagging, no weight is given to dataset, all have equal weights. The random component is the split, which considers a random subset (sqrt of num_features) and find its best split, thus the name random forest.
Bagging was also done on a limited dataset since training would take days and process would be killed for taking too much memory.

We took 100 trees at depth and found out the 1se values for different depths as shown

Depth	5	10	15	20
1se Values	85.3	91.7	93.6	96.6

Best value at depth 20 as we can see, so we choose it as our hyperparameter.
For different trees with depth 10 we have:

Num Trees	50	100	150
1se value	80.3	91.7	93.1

Although there is increase in accuracy from 100 trees to 150, paper chose 100 because the accuracy score did not vary much much time increased by over 70%(our analysis)

Accuracy on total dataset comes out to be : 98.2%

For limited Feature Set(hand)

We fixed number of trees 100 and changed depth parameter and their corresponding 1se values:

Depth	5	10	15	20
1se Values	81.8	87.9	90.5	93.1

There is an increase in accuracy with depth, and we considered 20 as depth

We fixed depth to be 10 and changed number of trees and their corresponding 1se values:

Num Trees	50	100	150
1se value	73.8	87.9	87.1

We can actually see there is a decrease in accuracy, with 150 rather than 100 , which is a sign of overfitting. Hence we choose 100 trees and depth 20 for both full and limited features.

Accuracy on total dataset for hand is : 93.3

Logistic regression

Logistic regression is done with total 4 types of data sets, whole data set, hand dataset, chest dataset, ankle dataset. For whole data set we are getting an accuracy of 99.8%, while for hand dataset we are getting at 80% accuracy, for chest we are getting at 84% accuracy, for ankle dataset we are getting at 82% accuracy. So with this we can infer that single chest data is giving more accuracy.

For each of the above datasets, l2 regularization is applied with parameter C, and using validation sets and accuracy we found that 0.01 is the best C value for which we will get high accuracies, so we calculated above accuracies for testing data using C value as 0.01.

We can see the different accuracies with change in C value in the following tables.

C values	0.001	0.01	0.1
Ankle accuracies	81.51%	81.72%	81.45%

We can also see the same data for the Chest dataset

C values	0.001	0.01	0.1
Chest accuracies	83.40%	84.15%	83.43%

We can also see the same data for the Hand dataset

C values	0.001	0.01	0.1
Hand accuracies	79.37%	80.62%	79.49%

The loss function used is as follows :

$$J(\theta) = \sum_{i=0}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{1}{C}||\theta||_2^2$$

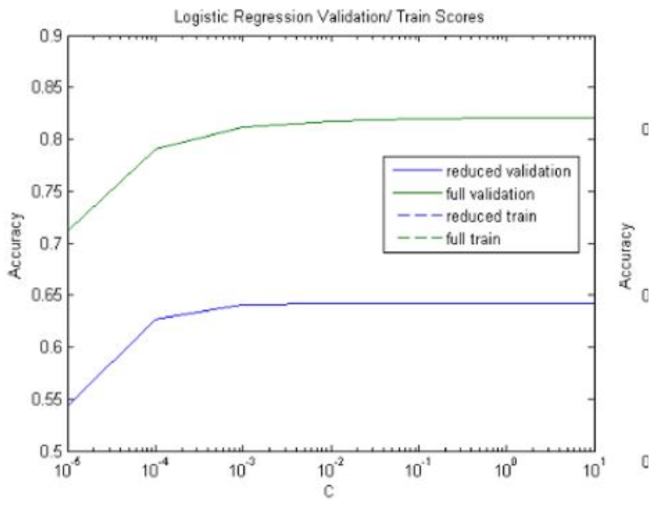
C is the parameter we discussed earlier which is l2 regularization constant, it is introduced to reduce overfitting of training data, if C value is too high then that l2 regularization term will vanish and it becomes overfitting and if C value is too low then l2 regularization becomes significant and it will be underfitting and after final testing and analysis for multiple datasets we came to conclusion that 0.01 is the best value for C which gives very low cost function value and high accuracy value.

We first divide the dataset into 80% training and 20% testing, and in this 80% training we do cross validation with 5 fold as mentioned and find the best C parameter value, and then we will find the accuracy for testing with that C parameter. We have used newton-cg as our gradient descent optimizer process, because it well suites for l2 regularization process and reaches to global minimum in less time.

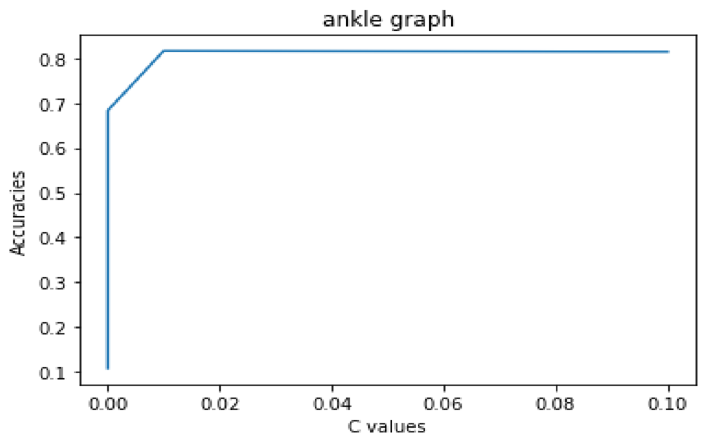
The following table represents accuracies for different datasets :

Dataset type	Accuracies
Whole dataset	95.78%
Hand dataset	80.62%
Chest dataset	84.15%
Ankle dataset	81.72%

The first 3 tables infer that 0.01 is the optimal C value which can be seen from the graph given in the report.



From the below graph we can say that C value maximizes at 0.01 same as above graph and tables.



As we can see in the aboev graphs it gets peak at 0.01 and then starts decreasing further.

<u>SVM (support vector machine)</u>

In this we have multiple classes to classify. The naive svm classifier is a binary classifier so we need to choose 2 one vs one classifiers or n one vs rest classifiers to classify into n different classes. The C parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassified more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.the main advantage if svm is that it can be non-linear and for testing we only need some points to consider (support vectors) as the value of c decreases the time to train increases since we need to consider the large margin and neglect the misclassifications

Here we used only three kernels for classification (linear , rbf and polynomial (3 degree)) which are commonly used and are reliable. Among these three rbf gives the highest accuracy @@@ and is robust the data is split into 80 percent training and 20 percent testing.as said earlier we classified by the mode of the classes in the results of the n choose 2 classes i,e majority

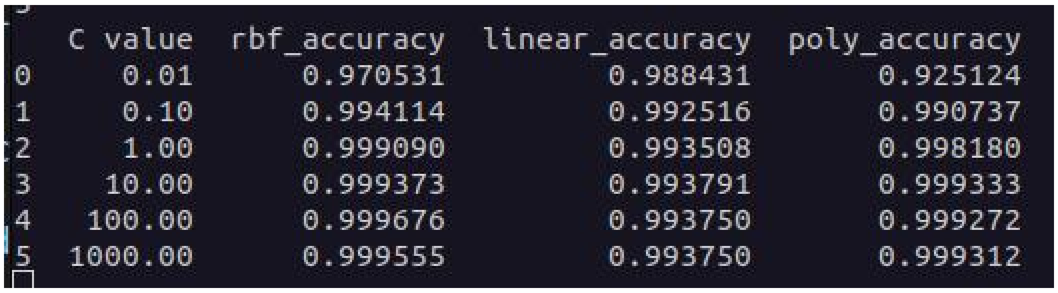
$$
\begin{aligned}
min_{y,w,b} \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\zeta_i \\
s.t.\ y^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta_i, \quad i = 1, ..., m \\
\zeta_i \geq 0, \quad i = 1, ..., m
\end{aligned}
$$

The above equation is the primal form of the equation which should be solved, which is a soft margin classifier which allows some misclassifications and tries to increase the margin as much as possible with limited misclassifications. The kernel functions project the data into higher dimensions so that we can classify them using the linear hyper planes there since all types of data cannot be classified optimally using only the hyperplanes. To select the optimal C and the kernel function we iterate through them
Also we can see that by normalizing the data the accuracy of the classifier increases. The normalization is done by removing the mean from the feature and dividing it by the standard deviation so that the variance of the data becomes equal to 1.
Svm is fairly well compared to the logistic regression and have fairly high accuracy but it took very high time to train the model and the testing complexity increases as the value of the c decreases

SVM seems to do a much better job in being able to distinguish between active and inactive tasks. And with the reduced features the accuracy decreases but with ankle and hand data is better than the chest data i,e the ankle and hand data are little important than that of the chest data.

We can see that the accuracies increases from 0.97 to 0.99 by increasing the value of c from 0.01 to 100 and we can see that the accuracy for the classifier using rbf is the best among the three

And the mean accuracy for the limited feature set is 0.96 and the mean accuracy for the all feature classification is 0.982



```
   C value  rbf_accuracy  linear_accuracy  poly_accuracy
0     0.01      0.970531         0.988431       0.925124
1     0.10      0.994114         0.992516       0.990737
2     1.00      0.999090         0.993508       0.998180
3    10.00      0.999373         0.993791       0.999333
4   100.00      0.999676         0.993750       0.999272
5  1000.00      0.999555         0.993750       0.999312
```