STAT 441 Final Report

Anusha Raisinghani
Jess Jiang
Reece Bajnathsingh

Time Created: 2023-04-24 01:48:12

Contents

1 Summary

This report aims to compare the predictive accuracy of three statistical classification methods on the task of recognizing human physical activities based on biometric data obtained from inertial measurement units (IMUs) and heart rate monitors worn by human subjects.

The three methods tested are: a Random Forest Classifier, a Stacked Classifier consisting of an ensemble of Elastic Net, Ridge and Support Vector Machine (SVM) classifiers, and finally, a Multi-Layer (Deep) Neural Network Classifier. For each classifier, k-fold cross validation was used to tune various hyperparameters via Grid Search. Following this, their accuracy, defined by

$$\frac{\text{num. of correct predictions}}{\text{total num. of predictions}} \tag{1}$$

, was measured by predicting activities in a test set. The results indicate that the Random Forest Classifier had an accuracy of 99.99%, the Stacked Classifier 87.54% and the Deep Neural Network Classifier 98.18%.

While the Random Forest classifier achieved the highest accuracy, it required more computationally and time intensive tuning and training compared to the Neural Network.

2 Introduction

Background Information

According to Gupta et al. (2022), Human Activity Recognition (HAR) can be defined as the art of identifying and naming activities using Artificial Intelligence from raw data collected via devices such as wearable sensors and smartphone inertial sensors. Human Activity Recognition has been a very attractive application of machine learning classification technique due to its many potential benefits in the healthcare industry. For instance, the research by Attal et al. (2015) mentions that activity recognition can provide aid to healthcare for children, the elderly and the disabled. They can be used to remotely monitor these groups' daily activities to detect abnormalities such as falls and prevent further potential risks. Gupta et al. (2022) also refers to numerous other applications such as physical activity tracking during sports and exercises to improve physical functions.

Dataset

The full dataset consists of physical activities including lying, sitting, standing, walking, running and so on, performed by 9 subjects (8 male and 1 female) wearing 3 inertial measurement units (IMUs) and a heart rate monitor (Reiss 2012). The three IMUs were placed over the wrist on the subject's dominant arm, on the chest and on the subject's dominant side ankle. The data was collected by having each of the subjects follow a protocol containing 12 activities. To briefly summarize, the protocol consists of performing a sequence of activities such as lying for 3 minutes, followed by sitting for 3 minutes and so on. The protocol is described in details in the dataset's corresponding README document.

Due to computational limitations, this report only considers a subset of two subjects from the original dataset: ID 101, and ID 108. Our dataset comprising the data of the two subjects contains 512059 observations. Each row of the data contains 54 columns. The columns are as follows:

- 1 timestamp (s)
- 2 activityID
- 3 heart rate (bpm)
- 4-20 IMU hand
- 21-37 IMU chest
- 38-54 IMU ankle

The IMU sensory data contains the following columns:

- 1 temperature (°C)

- 2-4 3D-acceleration data (ms^{-2}), scale: $\pm$16g, resolution: 13-bit
- 5-7 3D-acceleration data (ms^{-2}), scale: $\pm$6g, resolution: 13-bit
- 8-10 3D-gyroscope data (rad/s)
- 11-13 3D-magnetometer data (μT)
- 14-17 orientation

Classification Methods

Some of the most popular traditional machine learning methods for HAR are Support Vector Machines (SVM), Random Forests (RF) and K-Nearest Neighbour (kNN). However, in recent years, the state of the art deep learning techniques have also been implemented, since they were proven to be more resource-efficient and capable than the traditional methods (Tee et al. 2022). Nevertheless, this report aims to provide a comprehensive comparison between three different classification techniques and their predictive capabilities on The Physical Activity Monitoring Dataset (PAMAP2) (Reiss 2012).

The three classifiers we applied to the data include a Random Forest Classifier, a Stacked Classifier which consisted of an ensemble where the base models were Ridge Classifiers and Elastic Net Classifiers while the meta-classifier was a SVM and a Deep Neural Network, specifically, a Multi-layer Perceptron (MLP). For each model, appropriate preprocessing was applied and specific hyperparameters were tuned using Grid Search with K-fold cross validation using a dedicated training set. Finally, their performance was evaluated based on predictions made on a test set. Specifically, these models are compared based on the following metrics:

Let TP_k, FP_k, TN_k, FN_k represent the number of true positive predictions, false positive predictions, true negative predictions and false negative predictions for class k in a multiclass classification setting with K classes. That is $k \in \{1, \ldots, K\}$. Then:

$$\text{accuracy} = \sum_{k=1}^{K} \frac{(TP_k + TN_k)}{(TP_k + FP_k + TN_k + FN_k)}$$

And across an individual class, k:

$$\text{f1-score}_k = \frac{TP_k}{TP_k + (1/2) \times (FP_k + FN_k)}$$

$$\text{precision}_k = \frac{TP_k}{TP_k + FP_k}$$

$$\text{recall}_k = \frac{TP_k}{TP_k + FN_k}$$

3 Methods

3.1 Pre-processing

Due to computational limitations, the pre-processing for this dataset initially involves taking a subset of the full dataset. In this case, we took all the data for subjects 101 and 108. Further, based on the README of the dataset, rows with activity ID 0 have been removed and orientation columns have also been removed since they are invalid. Additionally, the `timestamp` column was also removed since this analysis does not consider the time series aspect of the data for reasons discussed later in this report in the Discussion section

Additionally, the data was standardized for the Stacked Classifier and Deep Neural Network, but not the Random Forest since they are invariant to monotone transformations of the feature variables. Standardization is a scaling technique to make the data scale-free by transforming the distribution of the data into the following format: $\frac{x-\mu}{\sigma}$, i.e., scaling the data to a 0 mean and unit variance (Buitinck et al. 2013). Only the

training data was used to estimate the mean and standard deviations to avoid data leakage. Standard scaling is a good choice because the marginal distribution of the majority of features is normal as seen in Figure 1.

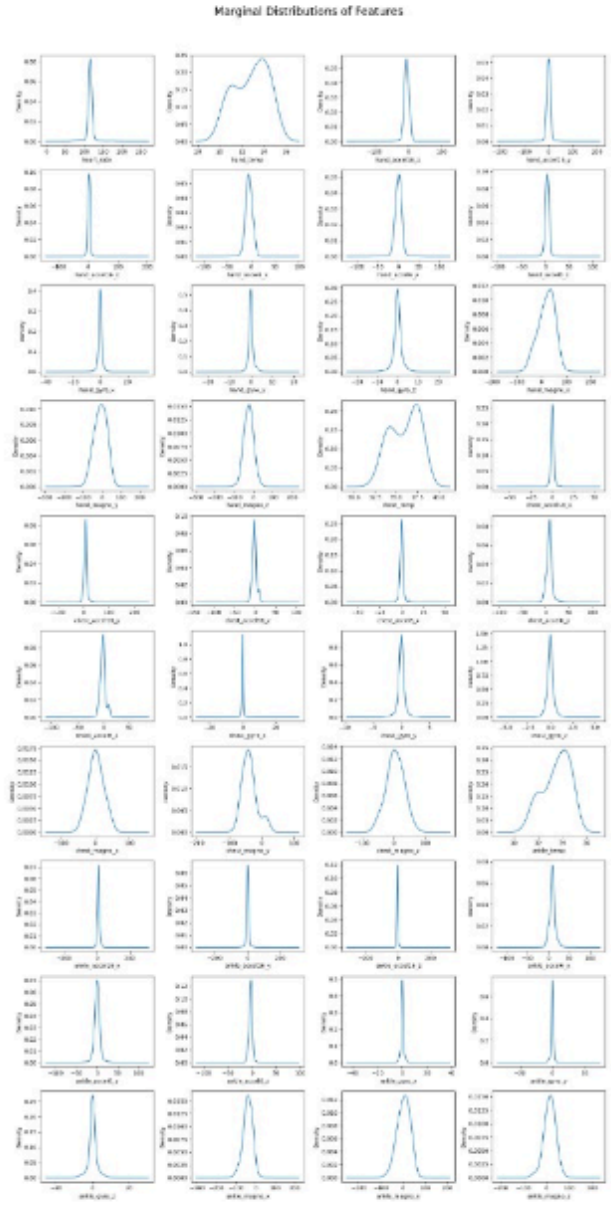


Figure 1: The figure shows the marginal distribution of features in the dataset

All the missing values in the data were found in numerical columns and these were imputed using the column means.

Finally, the preliminary analysis showed that the dataset classes were imbalanced and hence the Synthetic Minority Over- sampling Technique (SMOTE) library was used to balance the proportion of activities. SMOTE is an approach that over-samples the minority class to create "synthetic" examples rather than

over-sampling with replacement, and has proved quite effective in handwritten character recognition (Chawla et al. 2002).

3.2 Preliminary Analysis

In the initial analysis, the division of activities between the subjects was explored using stacked bar plots as shown in Figure 2. The plots clearly show that the dataset is imbalanced where the activities 'ascending stairs', 'descending stairs' and 'rope jumping' are minority classes. Hence, SMOTE was used to balance the classes.

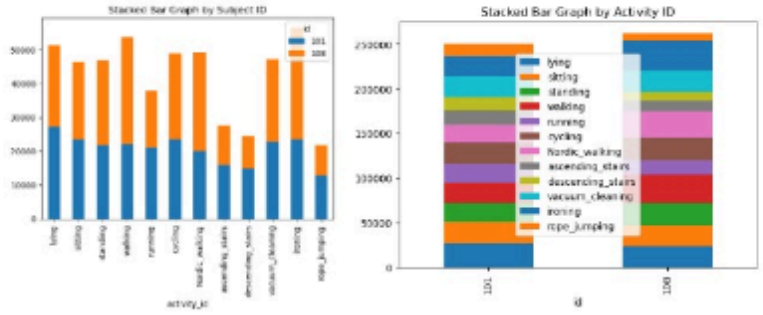


Figure 2: Left bar plot shows the count of observations of each activity separated by subjects and right bar plot shows count of observations for each subject separated by activities

Next, t-tests were conducted to compare the means of the biometric data, such as heart rate, hand temperature, etc. across activities and across subjects. The results for the t-test can be seen in Table 4. From the table it is evident that the p-values for equal means are 0, indicating that the means for different biometric features are not equal across activities and across subjects.

3.3 Random Forest Classifier

The first classifier trained was the Random Forest Classifier with 100 estimators.

Calibration Procedure

To tune the model, Coarse-to-Fine Hyperparameter Tuning with 3-fold cross validation was used. Coarse to Fine tuning combines the advantages of grid and random search (Scriven n.d.). In order to explore a relatively large search space of hyperparameters, firstly `RandomSearchCV` was used from Scikit-learn library followed by `GridSearchCV`. To do this, a dictionary specifying the grid of hyperparameter values was defined to be used as the search space. In the first call to `RandomSearchCV` (Buitinck et al. 2013), the number of iterations was specified to be 30 to limit the number of model runs. This means that 30 configurations of hyperparameters were formed by randomly sampling from the distributions outlined in the search space grid. The results of `RandomSearchCV` were then plotted in graphs of mean test error vs hyperparameter value to determine a narrowed range of values on which to perform an exhaustive grid search. Once the best performing hyperparameters were obtained, the Random Forest was fit on the whole train dataset.

The hyperparameters that were selected for tuning via grid search are:

- `n_features`: maximum number of features to consider at each split
- `max_depth`: maximum depth of the tree
- `min_samples_split`: minimum number of samples required to split a internal node
- `min_samples_leaf`: minimum number of samples required to be at a leaf node
- `bootstrap`: whether to use bootstrap samples when building the trees
- `loss`: whether to use the Gini Index or Entropy as the loss function

The random forest was trained on both SMOTE and non-SMOTE data. The best classifier was chosen based on its predictive accuracy when trained using each type of data as well as the time required to tune and fit it.

3.4 Stacked Classifier

The second classifier was the Stacked (Generalization) Classifier. This is an ensemble method that combines the strengths of multiple models. Stacking entails a multi-tier learning algorithm (usually just two-tiers) with multiple models in the first-tier (or the base level) and another classifier, called the meta classifier, that makes the final decision. While fitting the data, the base models learn from the data and output their respective predictions. The output of these models are called the meta-features and are passed as inputs to the meta-classifier for aggregation (Lee 2017). Stacked classifiers improve generalization of the model and reduces error rate by reducing the bias of the individual models used in the stack (Ravi 2020).

For the stacked classifier, an Elastic Net and a Ridge Classifier was used for the base level followed by a Linear Support Vector Machine (SVM) as the meta-classifier. Elastic Net is a penalized logistic regression model that includes both the L1 and L2 penalties during training. It aims to minimize the following loss function (Kargın):

$$L_{\text{enet}}(\hat{\beta}) = \frac{\Sigma_{i-1}^{n}(y_i - x_i'\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\Sigma_{j-1}^{m}\hat{\beta}_j^2 + \alpha\Sigma_{j-1}^{m}|\hat{\beta}_j|\right) \tag{2}$$

Here, $0 \leq \alpha \leq 1$ is a hyperparameter called ElasticNet mixing parameter. When $\alpha = 0$, the penalty is L2-penalty, and when $\alpha = 1$, the penalty is L1-penalty. This is the `l1_ratio` parameter in sklearn's ElasticNet class.

Calibration Procedure

For hyperparameter tuning, the Fine-to-Coarse Tuning method with 3-fold CV used to tune the Random Forest model was employed. However, tuning the stacked classifier can prove to be difficult due to the number of possible combinations with respect to the hyperparameters from all models in the stack (Ravi 2020). According to Ravi (2020), it is heuristic to tune and check the individual learners first and then to proceed implementing the stack by splitting the data into two parts - one for training the base-classifiers and another to train the meta-classifier. Then, Fine-to-Coarse tuning was used to find the optimal hyperparameters for the ridge classifier (α) and the elastic net classifier (α and `l1_ratio`). Finally, after obtaining the optimal hyperparameters for the base learners, Fine-to-Coarse tuning was used to tune the meta classifier.

For the heuristic method, a model was tuned and trained for both the SMOTE and non-SMOTE data, similar to random forests, to facilitate a comparison between which data does better.

3.5 Neural Network Classifier

The final classifier trained was a Neural Network Classifier. Specifically, a Multi-layer Network was trained with the following architecture:

- Linear layer(`in_features`=54, `out_features`=64)
- LeakyReLU activation layer
- Dropout(p)
- Linear layer(`in_features`=64, `out_features`=`num_classes`)
- LeakyReLU
- LogSoftmax(`dim`=1)

The loss function used to train the model was NLLLoss or Negative Log Likelihood loss (Paszke et al. 2019) This loss function is useful for solving multi-class classification problems and according to the Pytorch documentation, it expects as input, the log probability of each class for a given (feature, label) pair. Thus, LogSoftmax was used as the final layer in the neural network.

The optimizer used is the Adam optimizer with learning rate 0.001 which is a first-order gradient based optimization technique which is well suited for problems with large datasets (Kingma and Ba 2017).

Leaky ReLU is used as an activation function because it mitigates the "dying ReLU" problem. Since ReLU returns 0 for negative inputs, the gradient becomes 0 so the neuron stops learning when it becomes negative. This makes the network sensitive to weight initializations (Lu 2020). Leaky ReLU tries to solve this by providing a small gradient for negative inputs to allow recovery.

$$\text{Leaky ReLU} = \begin{cases} 0.01x, & x \leq 0 \\ x, & x > 0 \end{cases} \tag{3}$$

In the above architecture, the input to the Dropout layer, the dropout probability p, will be treated as a hyperparameter for tuning. Dropout is a mechanism where during training, some of the inputs to the dropout layer are randomly set to zero with probability p via sampling from a Bernoulli distribution. This has been shown to be helpful for regularization (Hinton et al. 2012).

Additionally, the batch size used to train the model will be treated as a hyperparameter. The batch size refers to the number of input observations that are sent through the network in a forward pass before back propagating and updating the network parameters.

Neural Network Pre-Processing

1. Data was loaded and preliminary preprocessing was executed as in section 3.1.
2. Sklearn's `LabelEncoder()` was used to encode the class labels in the data to be in the range [0, `num_classes`). This is done because the PyTorch implementation loss function used, NLLLoss, requires labels to be in that range. Additionally the new encoding was saved to a variable for decoding activity ids.
3. The `id` column is one-hot encoded. That is, the original `id` column was dropped and replaced with indicator columns `id_101` and `id_108`. For example, if the `id` of a subject in a particular row of data is 101 then `id_101` is set to 1 and `id_108` was set to 0.
4. The data is split into train/validation/test splits using an 60/20/20 split using sklearn's `train_test_split()` with the stratified option to ensure class proportions remain the same in each split.

Calibration Procedure to get Final Model

1. First, holdout cross validation was employed in order to train an initial model. That is a neural network was trained with initial batch size of 32 and dropout probability of 0.3 using the training data for 10 epochs using training data without applying SMOTE. After each epoch, the model's performance is validated against the validation set and if the average loss over batches decreases for that epoch, the new model was saved. Then, prediction metrics as well as a confusion matrix were plotted to gauge performance.
2. Step 1 was repeated using training data with SMOTE applied
3. The model that performs best on the test set in terms of our prediction metrics was used to determine whether we continue to use SMOTE or not.
4. GridSearchCV was used to do 3-fold cross validation on a parameter grid specifying batch sizes of (16,32,64) and dropout probabilities of (0,0.3) using the preferred dataset. The best performing model in terms of CV accuracy determines which hyperparameters were used to train the final model
5. A final neural network model was trained using a combination of the training and the validation set with the best hyperparameters found in Step 4. This model was then evaluated on the test set and the prediction metrics were used to compare performance with the other classifiers in this report.

4 Results

The classification reports produced by using the best trained classifier using each of the three methods: Random Forest, Stacked and Deep Neural Networks are shown in the tables below.

Table 1: Reported Classification Metrics for Random Forest on Non-SMOTE Data

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	1.000e+00	0.9995	0.9999	1.0000	0.9997	0.9996	1	0.9989	0.9986	1	0.9996	1.0000	0.9997	0.9996	0.9997
recall	9.997e-01	0.9999	0.9996	0.9996	0.9998	1.0000	1	0.9987	0.9988	1	1.0000	0.9993	0.9997	0.9996	0.9997
f1-score	9.998e-01	0.9997	0.9997	0.9998	0.9997	0.9998	1	0.9988	0.9987	1	0.9997	0.9997	0.9997	0.9996	0.9997
support	1.027e+04	9281.0000	9375.0000	10757.0000	7559.0000	9810.0000	9831	5515.0000	4911.0000	9447	11313.0000	4343.0000	0.9997	102412.0000	102412.0000

Table 2: Reported Classification Metrics for Stacked Classifier on non-SMOTE Data

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.807e-01	0.9502	0.9372	0.7698	0.8131	0.9795	0.7262	0.7537	0.6823	0.9236	0.9703	0.7921	0.8754	0.8565	0.8739
recall	9.688e-01	0.9750	0.9818	0.8402	0.8063	0.9684	0.8295	0.6647	0.7172	0.9431	0.9734	0.7868	0.8754	0.8554	0.8754
f1-score	9.748e-01	0.9625	0.9589	0.8035	0.8097	0.9739	0.6744	0.7064	0.6992	0.9332	0.9718	0.7894	0.8754	0.8552	0.8739
support	1.027e+04	9281.0000	9375.0000	10757.0000	7559.0000	9810.0000	9831.0000	5515.0000	4911.0000	9447.0000	11313.0000	4343.0000	0.8754	102412.0000	102412.0000

Table 3: Reported Classification Metrics for Final Neural Net

Unnamed: 0	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.994e-01	0.9895	0.9852	0.9950	0.9954	0.9994	0.9930	0.9701	0.9783	0.9988	0.9991	0.9874	0.9926	0.9909	0.9926
recall	9.946e-01	0.9939	0.9933	0.9972	0.9919	0.9987	0.9964	0.9811	0.9656	0.9969	0.9905	0.9924	0.9926	0.9911	0.9926
f1-score	9.970e-01	0.9917	0.9892	0.9961	0.9936	0.9990	0.9947	0.9756	0.9719	0.9979	0.9948	0.9899	0.9926	0.9910	0.9926
support	1.027e+04	9280.0000	9376.0000	10757.0000	7559.0000	9810.0000	9830.0000	5515.0000	4911.0000	9447.0000	11313.0000	4344.0000	0.9926	102412.0000	102412.0000

From the above results it can be seen that the best performing classifier was the Random Forest with an overall accuracy of 99.99%. The best Random Forest Classifier fit was the one that was trained on data that was not treated by SMOTE, but there was only a marginal increase in performance ($< 0.01\%$ increase in accuracy) over the SMOTE treated dataset. Due to the extra computation required to treat a dataset with SMOTE and also due to the increased size of the training data as a result of SMOTE, it was decided that it should not be considered for the best fitting model. Additionally, from Table 1, precision, recall and f1-score across classes are very similar and close to 1. Essentially, the Random Forest was able to provide almost perfect out of sample predictions. This is clearly highlighted by the Figure 3.

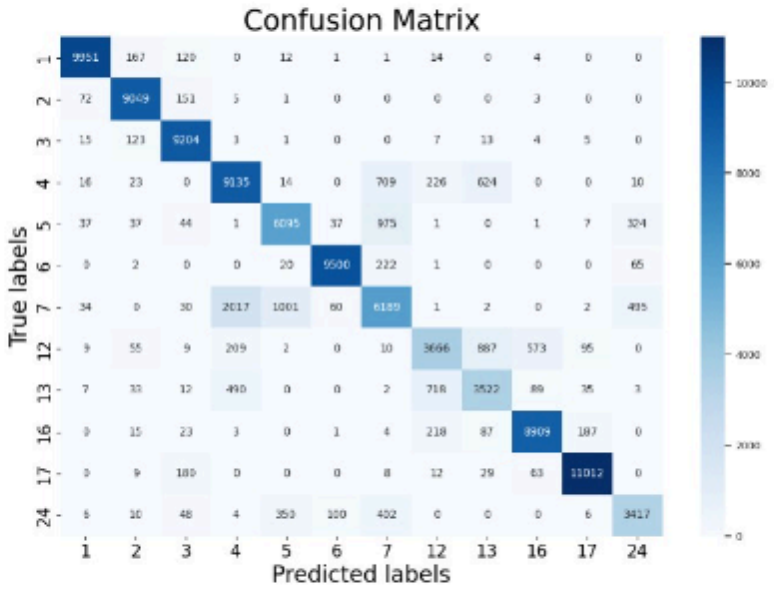


Figure 3: The figure shows the confusion matrix for the best fitting random forest

On the other hand, the Stacked Classifier which was trained on data treated without SMOTE exhibited an accuracy of 87.54% while the one trained with SMOTE showed an overall accuracy of 84.78%. For similar reasons as in the Random Forest, the Stacked Classifier without SMOTE was considered the better of the two. In fact the SMOTE Stacked Classifier had greatest difficulty classifying activities 7 and 24 according to their precision, recall and f1-scores. The Stacked Classifier without SMOTE saw an improvement in its ability to discern classes 7 and 24 suggesting that the additional data points introduced by SMOTE have likely obscured the decision boundaries between classes. This is because while generating new synthetic data points for the minority class, SMOTE does not consider nearby points that do not belong to the minority class and thus, may create overlapping regions between classes. The Stacked Classifier ensemble method consists of Ridge Classifiers, Elastic Net Classifiers and a Linear SVM which are all linear classifiers. Such a classifier would experience greater difficulty in separating classes that have complex and non-linear decision boundaries. This hypothesis is supported by the evidence that the Random Forest Classifier performed exceptionally well as Random Forests are suited to fitting non-linear decision boundaries.

Finally, the Neural Network Classifier which was fit on data not treated by SMOTE achieved an overall accuracy of 97.73 % on the test set. Examining the classification report more closely reveals that while it performs well generally in classifying all activities, it shows a relative difficulty in classifying classes 12, 13 and 24 which show f1-scores (harmonic mean between precision and recall) of 0.903, 0.901 and 0.9491 respectively. While these are still good, they are lower in comparison to the other classes which have scores between 0.97 and 0.99. It is also interesting to note that classes 12, 13 and 24 also have the least supporting points of all the classes. This suggests that their under-representation in the dataset may be affecting the neural network's ability to learn how to classify them properly. This hypothesis is somewhat supported by the results of the best fitting Neural Network Classifier trained using data treated using SMOTE. The accuracy raises to 98.18% and the f1-scores for the under-represented classes 12, 13, 24 rise slightly to 0.935, 0.929 and 0.958 respectively. The performance of the Neural Network is comparable to that of the Random Forest Classifier but still not as good. This is likely because it is also able to accommodate non-linearity.

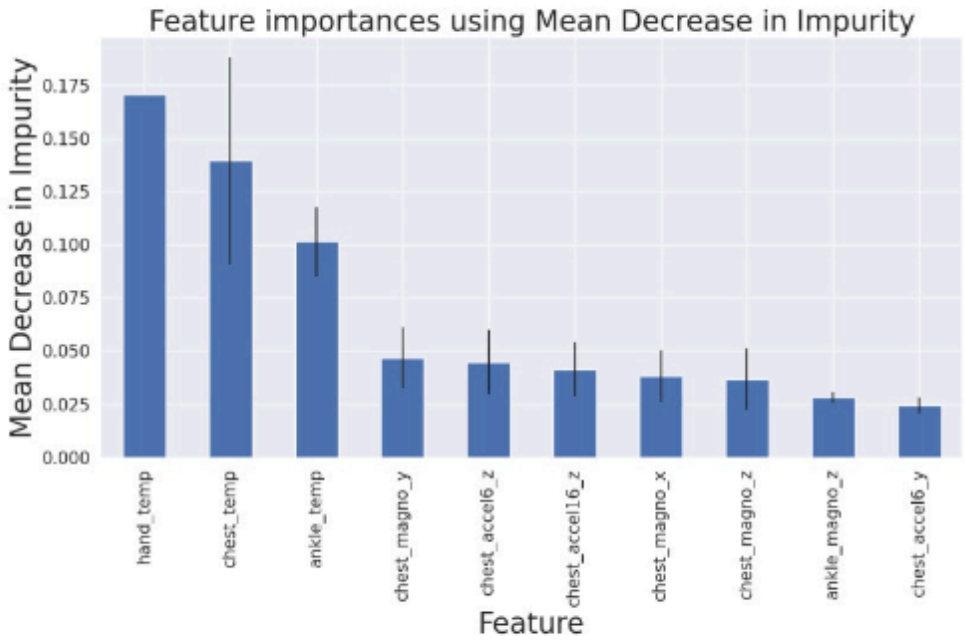


Figure 4: The figure feature importance in the best random forest classifier

In order to gain more insight into the reason that the Random Forests are performing the best, a bar plot of the feature importances of the ten most important features was examined as show in Figure 4. The plot shows that `hand_temp`, `chest_temp` and `ankle_temp` were by far the most important features used to classify the dataset as they provide the greatest mean decrease in Gini impurity. In a dataset with K classes, Gini impurity for some node m is defined as

$$\sum_{i=1}^{K} p_m(i) \times (1 - p_m(i)) \tag{4}$$

where $p_m(i)$ is the probability of picking a datapoint with class i from node m.

To more closely examine these features, a plot of the marginal distribution of each feature separated by activity ID was produced in Figure 5. These plots show that the top three features when separated by activity ID, follow distributions that do not significantly overlap. This is a good indicator that these features would do well as predictors for separating the data.

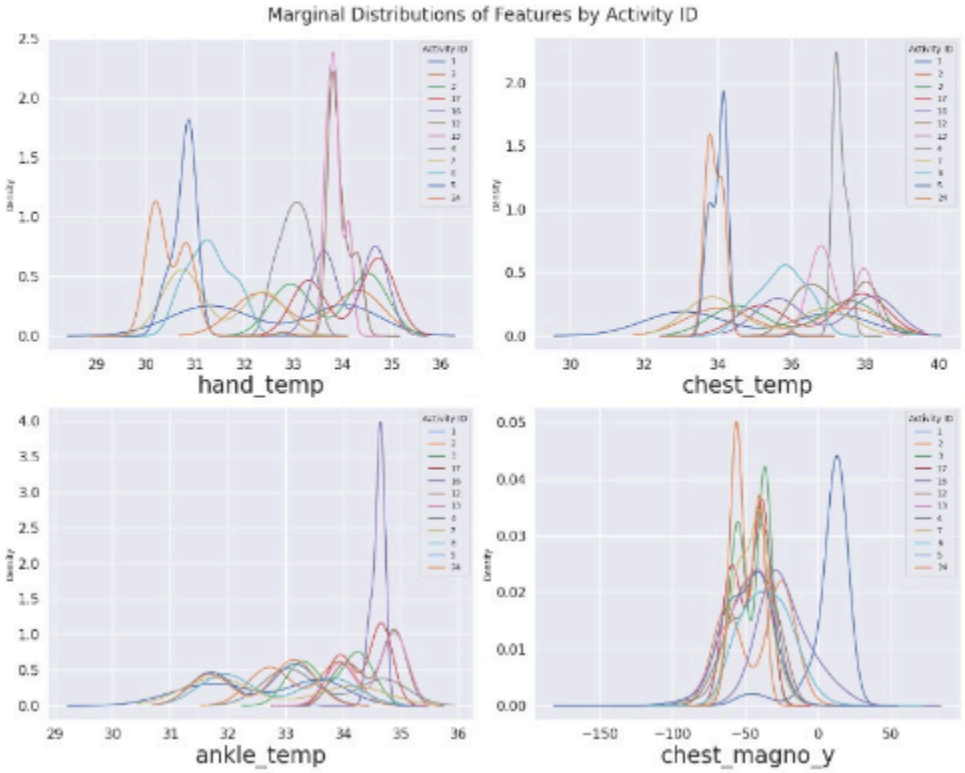


Figure 5: The figure shows marginal distribution of each feature separated by activity ID

Since no feature selection was done when fitting the Stacking Classifier and Neural Network Models, then it is possible that the additional 37 features used to model the data may have been redundant or caused the model to learn irrelevant relationships between the feature and target causing over fitting (Liu 2010).

In conclusion, with reference to the research question. The best predictive model was found to be the Random Forest Classifier. This is because the dataset has a large number of observations as well as the features `hand_temp`, `chest_temp` and `ankle_temp` which provide sufficient information about the class distributions to allow Random Forests to model the likely non-linear decision boundary between classes to produce very accurate out of sample predictions.

5 Discussion

From the previous results section, it can be seen that by this report's methods, a very high degree of accuracy, precision and recall were obtained from all three classifiers. The models in this report were primarily trained using Google Colab and the University of Waterloo's Compute Servers JupyterHub. Even with these limited computational resources,the methods in this paper were able to efficiently find optimal hyperparameter configurations for each model on a relatively large dataset. Particularly, the method of Random Grid Search cross validation to narrow the hyperparameter search space followed by a fixed Grid Search cross validation allowed exploration of the hyperparameter search space in an efficient manner. For the Deep Neural Network, employing hold out cross validation followed by a grid search cross validation also allowed evaluation of SMOTE vs non SMOTE datasets followed by hyperparameter optimization in an efficient manner.

Due to the large number of features in the dataset, KNN (k-Nearest Neighbour) Classifier was avoided. KNNs work well when the number of features are small but perform worse for larger features due to the Curse of

Dimensionality. As the number of features increases, points that may be similar may have larger distances, which would lead to inaccurate results (Kouiroukidis and Evangelidis 2011). Further, SVMs (Support Vector Machines) were avoided due to the high training time associated with such a large dataset.

A limitation of our analysis is that we were only able to train and classify on 2 out of the 9 subjects contained in the PAMAP2 dataset due to limited computational resources and time. This means that there would be less variation in the features seen in our analyzed dataset as compared to the full dataset, consequently reducing the complexity of the classification. As such, the test accuracy may be overestimated. That is, the accuracy of each classification method would likely be lower if they were to be trained and tested using all subjects in the dataset.

Many analyses of sensor data for HAR tend to make use of the time series aspect of the data to help improve the accuracy of predictions. This is done by leveraging architectures that are able to remember sequential information such as Long Short-Term Memory (LSTM) networks (Ahmad et al. 2019). Due to the nature of data collection for this dataset which involved having subjects perform activities according to a pre-defined protocol, training a classifier based on the time series would provide challenges. The classifier would not learn any generalizable patterns in the time series but rather the protocol sequence. Furthermore, due to the nature of the protocol, data is highly clustered by class making train/test splits very imbalanced in terms of class proportions. As such, this limited the feasibility of applying such methods for the analysis.

Future researchers who have sufficient computational resources and time may look to explore a similar classifier analysis using data from all 9 subjects as opposed to only 2. Furthermore, as the accuracy of these methods is expected to decrease on the larger dataset with more variation, 5-fold cross validation could be utilized in various parts of the analysis where this report used 3 folds to save on computation cost. Additionally the range of hyper parameters searched across using the grid searches could also be expanded to find even more optimal model configurations. Furthermore, feature selection methods such as Principal Component Analysis (PCA) may be employed to reduce the feature space and eliminate unnecessary predictors to improve the models.

References

Ahmad, W., Kazmi, B., and Ali, H. (2019), "Human activity recognition using multi-head CNN followed by LSTM," in *2019 15th international conference on emerging technologies (ICET)*, IEEE, pp. 1–6.

Attal, F., Mohammed, S., Dedabrishvili, M., Chamroukhi, F., Oukhellou, L., and Amirat, Y. (2015), "Physical human activity recognition using wearable sensors," *Sensors*, MDPI, 15, 31314–31338.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013), "API design for machine learning software: Experiences from the scikit-learn project," in *ECML PKDD workshop: Languages for data mining and machine learning*, pp. 108–122.

Chawla, N., Bowyer, K., Hall, L., and Kegelmeyer, W. (2002), "SMOTE: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 16, 321–357.

Gupta, N., Gupta, S. K., Pathak, R. K., Jain, V., Rashidi, P., and Suri, J. S. (2022), "Human activity recognition in artificial intelligence framework: A narrative review," *Artificial intelligence review*, Springer, 55, 4755–4808.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012), "Improving neural networks by preventing co-adaptation of feature detectors."

Kargın, K."ElasticNet Regression Fundamentals and Modeling in Python — medium.com," https://medium.com/mlearning-ai/elasticnet-regression-fundamentals-and-modeling-in-python-8668f3c2e39e.

Kingma, D. P., and Ba, J. (2017), "Adam: A method for stochastic optimization."

Kouiroukidis, N., and Evangelidis, G. (2011), "The effects of dimensionality curse in high dimensional kNN search," in *2011 15th panhellenic conference on informatics*, pp. 41–45. https://doi.org/10.1109/PCI.2011.45.

Lee, E. S. (2017), "Exploring the performance of stacking classifier to predict depression among the elderly," in *2017 IEEE international conference on healthcare informatics (ICHI)*, pp. 13–20. https://doi.org/10.1109/ICHI.2017.95.

Liu, H. (2010), "Feature selection," in *Encyclopedia of machine learning*, eds. C. Sammut and G. I. Webb, Boston, MA: Springer US, pp. 402–406. https://doi.org/10.1007/978-0-387-30164-8_306.

Lu, L. (2020), "Dying ReLU and initialization: Theory and numerical examples," *Communications in Computational Physics*, Global Science Press, 28, 1671–1706. https://doi.org/10.4208/cicp.oa-2020-0165.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019), "PyTorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems 32*, Curran Associates, Inc., pp. 8024–8035.

Ravi, A. (2020), "Stacked generalization for human activity recognition," *arXiv preprint arXiv:2009.10312*.

Reiss, A. (2012), "PAMAP2 physical activity monitoring data set."

Scriven, A. (n.d.). "Informed search: Coarse to fine."

Tee, W. Z., Dave, R., Seliya, J., and Vanamala, M. (2022), "A close look into human activity recognition models using deep learning," in *2022 3rd international conference on computing, networks and internet of things (CNIOT)*, IEEE, pp. 201–206.

6 Appendix

6.1 Setup

```
# if (install_packages){
#   conda_create(envname = "r-reticulate", conda = "auto")
#   use_condaenv(condaenv = "r-reticulate", conda = "auto")
#   packages = c("pandas==1.5.3 ",
#                "scikit-learn==1.1.2",
#                "matplotlib",
#                "imbalanced-learn",
#                "seaborn",
#                "numpy",
#                "scipy",
#                "skorch")
#   conda_update(conda = "auto")
#   py_install(packages = packages)
#
#   # Using pip to install pytorch since need to pass
#   # a specific option for compatibility
#   conda_install(
#     envname = "r-reticulate",
#     c("torch"),
#     forge = TRUE,
#     channel = character(),
#     pip = TRUE,
#     pip_options = c("--index-url https://download.pytorch.org/whl/cu118"),
#     pip_ignore_installed = FALSE,
#     conda = "auto",
#     python_version = NULL,
#   )
# } else {
#   # Replace the envname appropriately for your system
#   conda_python(envname = "C:/Users/Reece/Documents/.conda/envs/r-reticulate",
#                conda = "auto", all = FALSE)
# }
```

6.2 Pre-processing

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import pickle
import os
import time
import torch
import random
import sklearn
```

```
from inspect import getsourcefile

# Setup for python code
```

```
# If True, evaluates code such as preprocessing, cv - takes a long time
# If False, loads variables from existing pickle files
evaluate = False

# Code to create a directory for storing pickle files
# Directory of Rmd file
file_path = os.path.abspath(getsourcefile(lambda:0))
dir_path = os.path.dirname(os.path.realpath(file_path))

# checking if the directory exists
if not os.path.exists(dir_path + '/pickle'):
    # if the directory is not present, create it
    os.makedirs(dir_path + '/pickle')

# checking if the directory exists
if not os.path.exists(dir_path + '/plot_images'):
    # if the directory is not present, create it
    os.makedirs(dir_path + '/plot_images')
```

6.3 Data Loading/Preprocessing

```
# Activity dict to map activity numbers to text actions
# Obtained from data readme
activity_map = {0: 'transient',\
                1: 'lying',\
                2: 'sitting',\
                3: 'standing',\
                4: 'walking',\
                5: 'running',\
                6: 'cycling',\
                7: 'Nordic_walking',\
                9: 'watching_TV',\
                10: 'computer_work',\
                11: 'car driving',\
                12: 'ascending_stairs',\
                13: 'descending_stairs',\
                16: 'vacuum_cleaning',\
                17: 'ironing',\
                18: 'folding_laundry',\
                19: 'house_cleaning',\
                20: 'playing_soccer',\
                24: 'rope_jumping' }

# file name root to load data from
file_root = 'PAMAP2_Dataset/Protocol/'

# List of Subject ids we want to load data from
# The full data has subjects with id 101 to 109
subject_ids = [str(i) for i in [101, 108]]
```

```
def build_names(part):
  '''
  Builds the column names for the PAMAP2 dataset for a given
```

```
    body part

    Parameters:
      part : str
        the body part name to generate column names from

    Returns:
      list[str]
        A list of column names
    '''
    cols = [part+'_temp']
    coords_3d = ['x','y','z']
    coords_4d = ['x','y','z','w']
    accel16 = [part+'_accel16_'+coord for coord in coords_3d]
    accel6 = [part+'_accel6_'+coord for coord in coords_3d]
    gyro = [part+'_gyro_'+coord for coord in coords_3d]
    magno = [part+'_magno_'+coord for coord in coords_3d]
    orient = [part+'_orient_'+coord for coord in coords_4d]
    cols.extend(accel16+accel6+gyro+magno+orient)
    return cols

def generate_col_names():
    '''
    Builds all the column names for the PAMAP2 dataset

    Parameters:
      None

    Returns:
      list[str]
        A list of all column names for the PAMAP2 dataset
    '''
    # First 3 cols according to readme
    col_names = ['time_stamp','activity_id','heart_rate']
    # Body parts with IMUs
    parts = ['hand','chest','ankle']
    for part in parts:
      col_names.extend(build_names(part))
    return col_names
```

```
def load_data(file_root, subject_ids, subset_size=0):
    '''
    Loads data for the subects corresponding to the given subject_ids from the
    PAMAP2 dataset located at the given file_root into a single pandas
    dataframe

    Parameters:

      file_root: str
        the path to the directory containing subject data from the
        PAMAP2 dataset

      subject_ids : list[str]
        a list of subject ids whose data is to be loaded
```

```
    subset_size: int
      if 0, loads all the data, else loads a random sample of size
      subset_size from the full dataset

  Returns:
    pd.DataFrame
      A pandas data frame of subject data from the PAMAP2 dataset
  '''
  data = pd.DataFrame()
  col_names = generate_col_names()
  for id in subject_ids:
    file_name = file_root + 'subject' + id +'.dat'
    subject_data = pd.read_table(file_name, header=None, sep='\\s+')

    # Columns in the raw data are labelled with numbers, here we rename
    # them with their text attribute names
    subject_data.columns = col_names
    # Include a column to indicate the subject's id in the merged
    # data frame
    subject_data['id'] = id
    data = pd.concat([data, subject_data], ignore_index=True)

  if subset_size > 0:
    data = data.sample(subset_size)
  return data
```

```
if evaluate:
  data = load_data(file_root=file_root, subject_ids=subject_ids)
```

```
if evaluate:
  with open('./pickle/data.pickle', 'wb') as f:
        pickle.dump(data, f)
  print("Data saved in ./pickle/data.pickle")
```

```
if not evaluate:
  with open('./pickle/data.pickle', 'rb') as f:
      data = pickle.load(f)
  print("Data loaded from ./pickle/data.pickle")
```

```
## Data loaded from ./pickle/data.pickle
```

```
# Check for missing data
data.isnull().sum()
```

```
## time_stamp             0
## activity_id            0
## heart_rate        465358
## hand_temp           2364
## hand_accel16_x      2364
## hand_accel16_y      2364
## hand_accel16_z      2364
## hand_accel6_x       2364
## hand_accel6_y       2364
## hand_accel6_z       2364
## hand_gyro_x         2364
```

```
## hand_gyro_y          2364
## hand_gyro_z          2364
## hand_magno_x         2364
## hand_magno_y         2364
## hand_magno_z         2364
## hand_orient_x        2364
## hand_orient_y        2364
## hand_orient_z        2364
## hand_orient_w        2364
## chest_temp           1172
## chest_accel16_x      1172
## chest_accel16_y      1172
## chest_accel16_z      1172
## chest_accel6_x       1172
## chest_accel6_y       1172
## chest_accel6_z       1172
## chest_gyro_x         1172
## chest_gyro_y         1172
## chest_gyro_z         1172
## chest_magno_x        1172
## chest_magno_y        1172
## chest_magno_z        1172
## chest_orient_x       1172
## chest_orient_y       1172
## chest_orient_z       1172
## chest_orient_w       1172
## ankle_temp           2286
## ankle_accel16_x      2286
## ankle_accel16_y      2286
## ankle_accel16_z      2286
## ankle_accel6_x       2286
## ankle_accel6_y       2286
## ankle_accel6_z       2286
## ankle_gyro_x         2286
## ankle_gyro_y         2286
## ankle_gyro_z         2286
## ankle_magno_x        2286
## ankle_magno_y        2286
## ankle_magno_z        2286
## ankle_orient_x       2286
## ankle_orient_y       2286
## ankle_orient_z       2286
## ankle_orient_w       2286
## id                      0
## dtype: int64
```

```
def preprocess(data):
  '''
  Performs preprocessing on given PAMAP2 data to impute missing values with the mean
  and removeactivity_id=0 and cols with 'orient' in its name

  Parameters:

    data: pd.DataFrame
```

```
      the path to the directory containing subject data from the
      PAMAP2 dataset

  Returns:
    pd.DataFrame
      A preprocessed pandas data frame of subject data from the PAMAP2
      dataset
  '''
  # Drop cols with orient since they are invalid for this dataset
  data = data.drop([col for col in data.columns if 'orient' in col], axis=1)

  for col in data.columns:
    # Impute columns with mean
    data[col] = data[col].fillna(data[col].mean())
  # Remove activities with id==0
  data = data.drop(data[data['activity_id']==0].index)

  # Convert subject ID to type int
  data['id'] = data['id'].astype(int)
  return data
```

```
if evaluate:
  data = preprocess(data)
  print("Preprocessing Complete")
```

```
if evaluate:
  with open('./pickle/data_pp.pickle', 'wb') as f:
        pickle.dump(data, f)
  print("Preprocessed Data saved in ./pickle/data_pp.pickle")
```

```
if not evaluate:
  with open('./pickle/data_pp.pickle', 'rb') as f:
      data = pickle.load(f)
  print("Preprocessed data loaded from ./pickle/data_pp.pickle")
```

```
## Preprocessed data loaded from ./pickle/data_pp.pickle
```

```
# Ensure no missing data
any(data.isnull().sum())
```

```
## False
```

```
# Summary of data
data.describe()
```

```
##           time_stamp    activity_id  ...  ankle_magno_z             id
## count  512059.000000  512059.000000  ...  512059.000000  512059.000000
## mean     1771.737644       8.233227  ...      14.879838     104.583013
## std      1131.048623       6.403413  ...      24.246808       3.499019
## min        37.660000       1.000000  ...    -102.716000     101.000000
## 25%       724.060000       3.000000  ...      -0.989966     101.000000
## 50%      1627.200000       6.000000  ...      16.003300     108.000000
## 75%      2866.900000      13.000000  ...      31.481000     108.000000
## max      3888.410000      24.000000  ...     122.521000     108.000000
##
## [8 rows x 43 columns]
```

```
# Plot marginal distributions of features
if evaluate:
  fig, ax = plt.subplots(figsize=(15,30), nrows=10, ncols=4,sharey=False)
  i = -1
  feature_cols = [col for col in data.columns if col not in ['time_stamp', 'activity_id', 'id']]
  for j, col in enumerate(feature_cols):
    if j % 4 == 0:
      i += 1
    data[col].plot.density(ax=ax[i][j%4], bw_method=0.5)
    ax[i][j%4].set_xlabel(col)

  fig.suptitle("Marginal Distributions of Features", fontsize=20, y=0.99)
  fig.tight_layout()
  fig.subplots_adjust(top=0.95)
  plt.savefig('./plot_images/feature_marginals.png')
  plt.show()
```

```
def plot_cv_results(param_grid, cv_results, title, height, width):
    '''
    Uses cv_results from GridSearchCV or RandomizedSearchCV and
    plots results on a graph
    Parameters:
    param_grid: dict
        The parameter grid used to execute the GridSearchCV or RandomSearchCV
        which produced the provided cv_results
    cv_results: pd.DataFrame
        the results from the cv_results_ attribute of the RandomSearchCV or
        GridSearchCV object fit to the training data
    title: str
        the figure title
    height: float
        the figure height (in inches)
    width: float
        the figure width (in inches)
    Returns:
        None
    '''

    fig, axes = plt.subplots(len(param_grid), 2,
                             figsize = (width, height),
                             sharey='row')
    for i in range(len(param_grid)):
        if len(param_grid) == 1:
            axes[0].set_ylabel("Mean Test Score", fontsize=25)
            axes[1].set_ylabel("Test Score", fontsize=25)
        else:
            axes[i,0].set_ylabel("Mean Test Score", fontsize=25)
            axes[1,1].set_ylabel("Test Score", fontsize=25)
    for i, (param, param_range) in enumerate(param_grid.items()):
        agg_df = cv_results.groupby(f'param_{param}')['mean_test_score']\
            .agg(mean_test_score = 'mean')
        if len(param_grid) == 1:
            axes[0].set_xlabel(param, fontsize=20)
            axes[1].set_xlabel(param, fontsize=20)
```

```
            axes[0].plot(agg_df.index, agg_df['mean_test_score'])
            axes[1].scatter(x=cv_results['param_'+ param], y=cv_results['mean_test_score'])
            axes[0].tick_params(axis='both', which='major', labelsize=15)
            axes[1].tick_params(axis='both', which='major', labelsize=15)
        else:
            axes[i,0].set_xlabel(param, fontsize=20)
            axes[i,1].set_xlabel(param, fontsize=20)
            axes[i,0].plot(agg_df.index, agg_df['mean_test_score'])
            axes[i,1].scatter(x=cv_results['param_'+ param], y=cv_results['mean_test_score'])
            axes[i,0].tick_params(axis='both', which='major', labelsize=15)
            axes[i,1].tick_params(axis='both', which='major', labelsize=15)
    fig.suptitle(title, fontsize=35, y=0.98)
    fig.tight_layout()
    fig.subplots_adjust(top=0.95)
    plt.show()
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

def split_data(data, omit_cols=['time_stamp','id'], omit_acts=None, test_frac=0.2,\
stratify=False, scale=False):
  # Omit particular actions. Eg. if omit_acts = [1,2], omit rows
  # with activity_id==1 and activity_id==2
  if omit_acts:
    data = data[-data.activity_id.isin(omit_acts)]

  # Drop cols specified in omit_cols
  data = data.drop(omit_cols, axis=1)

  # Split into features (X) and labels (y)
  X = data.drop(['activity_id'], axis=1)
  y = data['activity_id']

  # Setting stratify==True means that the samples produced have
  # class proportions representative of the original class proportions
  # in the unsplit data
  if stratify:
     X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_frac, random_state=441,stratify=y)
  else:
    X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=test_frac, random_state=441,stratify=None)
  if scale:
    scaler = StandardScaler()
    scaler.fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)
  return X_train, X_test, y_train, y_test
```

```
if evaluate:
  X_train, X_test, y_train, y_test = split_data(data, stratify=True)
  print("Successfully split data into train and test sets")
```

```
if evaluate:
  with open('./pickle/train_test.pickle', 'wb') as f:
      pickle.dump([X_train, X_test, y_train, y_test], f)
  print("Saved train and test data to pickle file ./pickle/train_test.pickle")
```

```
if not evaluate:
  with open('./pickle/train_test.pickle', 'rb') as f:
      X_train, X_test, y_train, y_test = pickle.load(f)
  print("Loaded train and test data from pickle file ./pickle/train_test.pickle")
```

```
## Loaded train and test data from pickle file ./pickle/train_test.pickle
```

6.3.1 Applying SMOTE

```
from imblearn.over_sampling import SMOTE

if evaluate:
  # Optional oversampling with SMOTE
  sm = SMOTE()
  X_smote, y_smote = sm.fit_resample(X_train, y_train)
  print("Sucessfully produced SMOTE datasets")
```

```
if evaluate:
  with open('./pickle/smote_split.pickle', 'wb') as f:
      pickle.dump((X_smote, y_smote), f)
  print("Successfully saved SMOTE datasets to pickle files")
```

```
if not evaluate:
  with open('./pickle/smote_split.pickle', 'rb') as f:
    (X_smote, y_smote) = pickle.load(f)
  print("Successfully loaded SMOTE datasets from pickle files")
```

```
## Successfully loaded SMOTE datasets from pickle files
```

6.4 Preliminary Analysis

6.4.1 T-test

```
from scipy.stats import f_oneway

sub_statistic, sub_pval = f_oneway(
 *(data.loc[data['id']==group, ~data.columns.isin(['time_stamp', 'activity_id', 'id'])]
 for group in data['id'].unique())
 )

activity_statistic, activity_pval = f_oneway(
 *(data.loc[data['activity_id']==group, ~data.columns.isin(['time_stamp', 'activity_id', 'id'])]
 for group in data['activity_id'].unique())
 )

biometric = [i for i in data.columns.to_list() if i not in ['time_stamp', 'activity_id', 'id']]

f_test_results = {'Between Subjects Statistic':sub_statistic,\
'Between Subjects p-value' :sub_pval,\
'Between Activities Statistic':activity_statistic,\
```

Table 4: Table Displaying t-test Results

	Between Subjects Statistic	Between Subjects p-value	Between Activities Statistic	Between Activities p-value
heart_rate	2050.9297	0.0000	3665.5593	0.000
hand_temp	35121.1448	0.0000	131602.2082	0.000
hand_accel16_x	3168.4708	0.0000	25210.2015	0.000
hand_accel16_y	307173.9887	0.0000	1709.7647	0.000
hand_accel16_z	1307.0099	0.0000	16723.0354	0.000
hand_accel6_x	2965.7092	0.0000	25082.2636	0.000
hand_accel6_y	310754.2826	0.0000	1712.8733	0.000
hand_accel6_z	872.1071	0.0000	17128.1559	0.000
hand_gyro_x	1903.9869	0.0000	519.2136	0.000
hand_gyro_y	11.5922	0.0007	1368.2195	0.000
hand_gyro_z	83.1220	0.0000	65.0982	0.000
hand_magno_x	1935.2959	0.0000	30409.3115	0.000
hand_magno_y	267028.7613	0.0000	4299.9642	0.000
hand_magno_z	3741.0045	0.0000	14920.6226	0.000
chest_temp	124718.6126	0.0000	33359.9922	0.000
chest_accel16_x	949.6633	0.0000	2436.7229	0.000
chest_accel16_y	925.8193	0.0000	12425.4129	0.000
chest_accel16_z	17357.7668	0.0000	77133.6292	0.000
chest_accel6_x	1538.5349	0.0000	2457.1802	0.000
chest_accel6_y	831.1609	0.0000	13374.7163	0.000
chest_accel6_z	19992.8055	0.0000	77489.9184	0.000
chest_gyro_x	0.9516	0.3293	46.6703	0.000
chest_gyro_y	65.2474	0.0000	3088.4889	0.000
chest_gyro_z	0.5721	0.4494	535.4684	0.000
chest_magno_x	4066.9070	0.0000	18914.8791	0.000
chest_magno_y	55931.4730	0.0000	66451.9501	0.000
chest_magno_z	3486.6649	0.0000	46462.8057	0.000
ankle_temp	11313.7382	0.0000	51944.3978	0.000
ankle_accel16_x	207.1160	0.0000	13131.8014	0.000
ankle_accel16_y	3303.3341	0.0000	791.4297	0.000
ankle_accel16_z	4450.4385	0.0000	7206.6412	0.000
ankle_accel6_x	199.6058	0.0000	16671.0311	0.000
ankle_accel6_y	4354.7847	0.0000	1021.4889	0.000
ankle_accel6_z	6339.9601	0.0000	11209.1303	0.000
ankle_gyro_x	201.7103	0.0000	656.0513	0.000
ankle_gyro_y	197.7874	0.0000	61.0693	0.000
ankle_gyro_z	49.7365	0.0000	1.6082	0.089
ankle_magno_x	109274.9857	0.0000	11576.4230	0.000
ankle_magno_y	29581.5779	0.0000	7033.6499	0.000
ankle_magno_z	1526.5429	0.0000	2220.2011	0.000

```
'Between Activities p-value':activity_pval}

stat_test_df = pd.DataFrame(f_test_results,index=biometric)
```

```
knitr::kable(py$stat_test_df, booktabs = TRUE, digits=4,
             caption="Table Displaying t-test Results") %>%
  kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

6.4.2 Visualisations

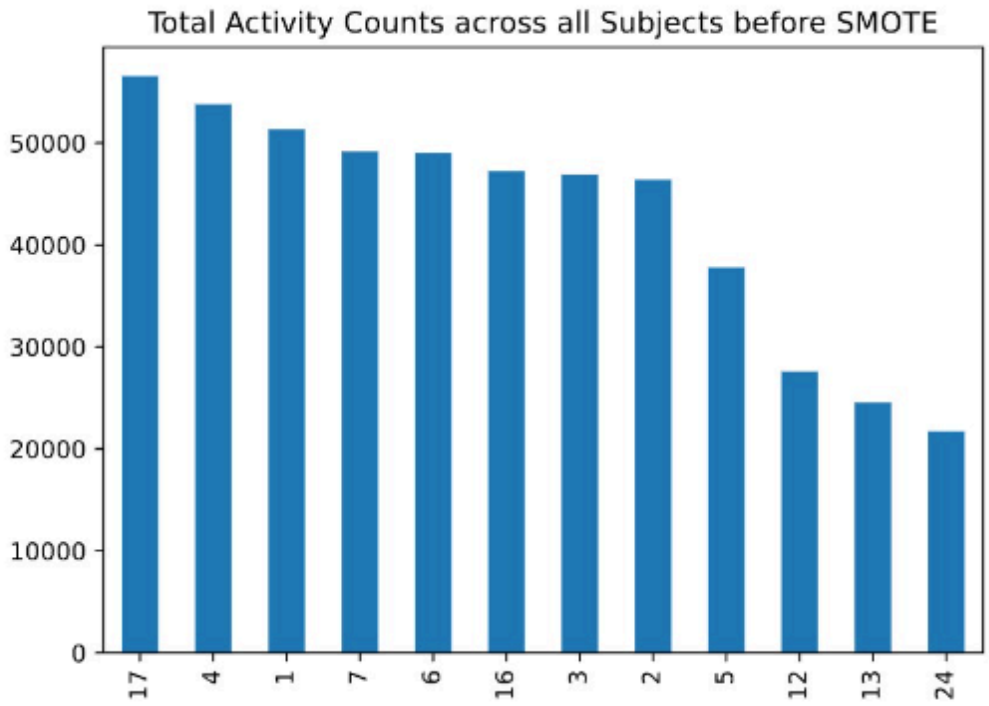
Stacked Bar Plot

```
def generate_stacked_plot(data, filename, stacked_by='subject_id'):
    '''
    Generates a stacked bar plot for either each subject ID stacked by activity ID or each activity
```

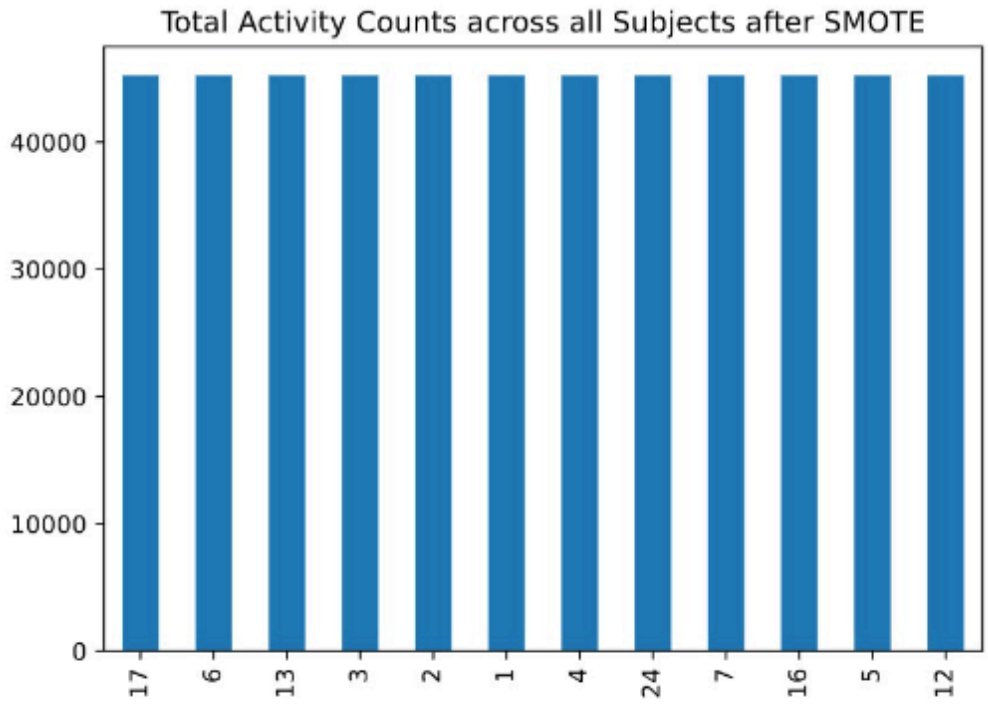
```
    ID stacked by subject ID.

    Parameters:
      data : pd.DataFrame
        The dataframe
      stacked_by : str
        Either 'subject_id' or 'activity_id' highlighting what each bar stacks on
    Returns:
      None
    '''
    grouped_count = data[['activity_id', 'id']]\
    .groupby(['activity_id', 'id']).size().reset_index(name='size')
    activity_labels = [activity_map[i] for i in list(grouped_count['activity_id'].unique()) ]
    if stacked_by == 'subject_id':
        grouped_count = grouped_count.pivot_table('size', ['activity_id'], 'id').reset_index()
        grouped_count.plot(x='activity_id', kind='bar', stacked=True,
        title='Stacked Bar Graph by Subject ID')
        plt.xticks(range(0, len(grouped_count['activity_id'].unique())), \
        activity_labels, rotation='vertical')
        plt.savefig('./plot_images/' + filename, bbox_inches = "tight")
    else:
        grouped_count = grouped_count.pivot_table('size', ['id'], 'activity_id').reset_index()
        grouped_count.plot(x='id', kind='bar', stacked=True,
        title='Stacked Bar Graph by Activity ID')
        plt.legend(labels=activity_labels, loc='center')
        plt.savefig('./plot_images/' + filename, bbox_inches = "tight")
```

```
generate_stacked_plot(data, filename='activity_plot.png')
generate_stacked_plot(data, filename='subject_plot.png', stacked_by='activity_id')
```

```
# Checking overall class distributions
data['activity_id'].value_counts().plot(kind = "bar", \
title='Total Activity Counts across all Subjects before SMOTE')
plt.show()
```




```
# Checking new class distributions
y_smote.value_counts().plot(kind = "bar",\
title='Total Activity Counts across all Subjects after SMOTE')
plt.show()
```



6.5 Random Forest

6.5.1 Randomized Search on Non-Smote Data

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

# Random search grid
rand_grid = {'max_features': ['sqrt', 'log2']\
             , 'max_depth': [x for x in range(50, 210, 10)]\
             , 'min_samples_split': [2, 5, 10]\
             , 'min_samples_leaf': [1, 2, 5]\
             , 'bootstrap': [True, False]\
             , 'criterion' :['gini', 'entropy']}

# Instantiate Random Forest Classifier
rf = RandomForestClassifier(n_jobs=100, n_estimators=100)

# Find optimal hyperparameters via Randomized Search through
# the defined rand_grid using Cross Validation
if evaluate:
  rand_search_rf = RandomizedSearchCV(estimator = rf, param_distributions = rand_grid, \
  n_iter = 30, cv = 3, verbose=2, random_state=441, return_train_score=True)
  rand_search_rf.fit(X_train, y_train)
  print("RandomSearchCV fit to train data complete")
  with open('./pickle/rand_search_rf.pickle', 'wb') as f:
```

```
    pickle.dump(rand_search_rf, f)

with open('./pickle/rand_search_rf.pickle', 'rb') as f:
  rand_search_rf = pickle.load(f)
```

```
def get_gridsearch_time(grid_search):
  ''' Takes a fitted grid_search object and prints the time taken for the fit'''
  mean_fit_time= grid_search.cv_results_['mean_fit_time']
  mean_score_time= grid_search.cv_results_['mean_score_time']
  n_splits  = grid_search.n_splits_ # number of cv splits
  n_iter = pd.DataFrame(grid_search.cv_results_).shape[0] # num iterations per split
  print('GridSearchCV Time Elapsed(s):', np.mean(mean_fit_time + mean_score_time) * n_splits * n_iter)
```

```
get_gridsearch_time(rand_search_rf)
```

```
## GridSearchCV Time Elapsed(s): 728.0432336330414
```

```
rand_search_cv_result = pd.DataFrame(rand_search_rf.cv_results_).sort_values(\
by='rank_test_score', ascending=True)
plot_cv_results(rand_grid, rand_search_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=20, width=20)
```

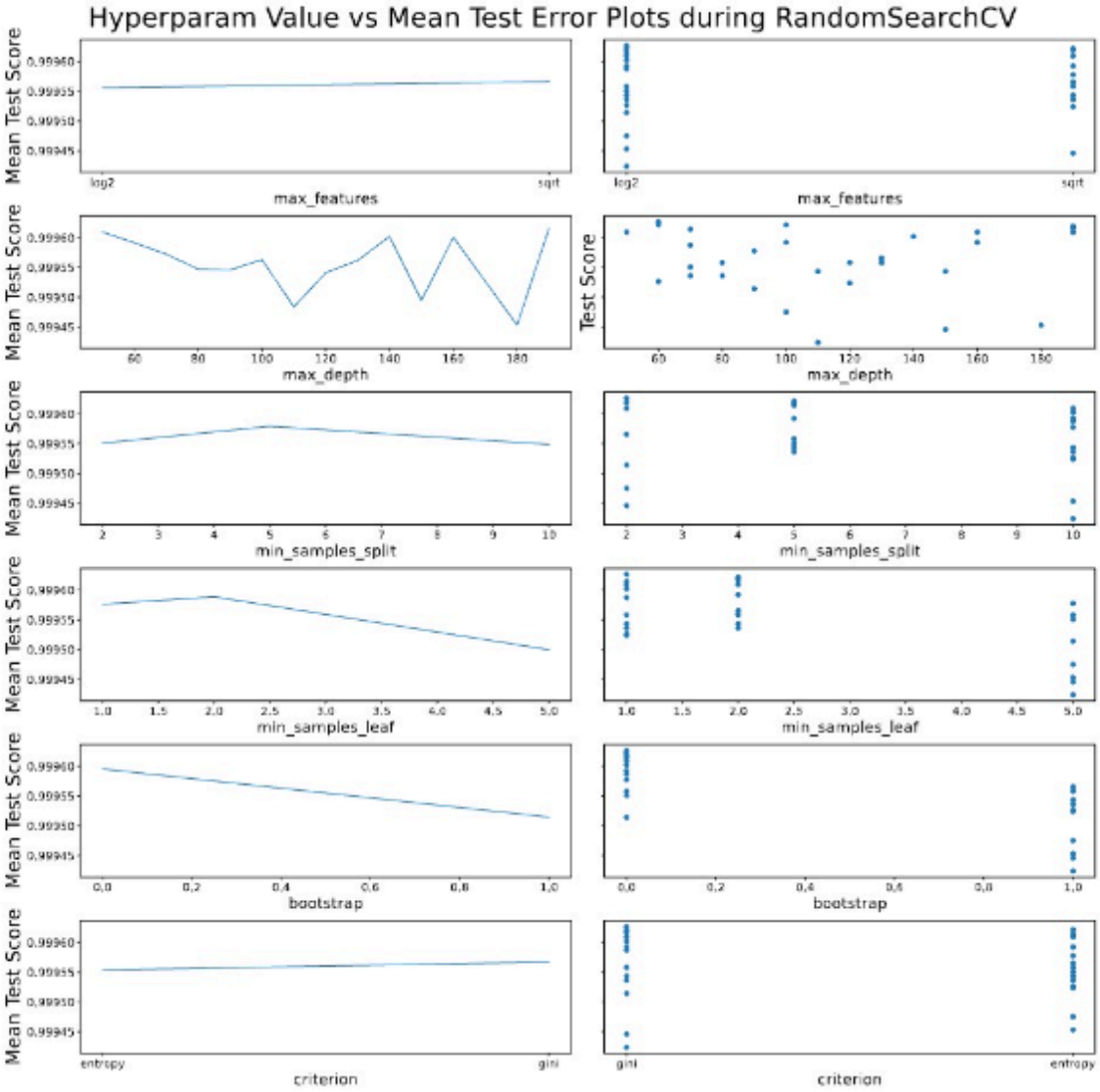


Figure 6: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV on Random Forest with Non-Smote Data

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'bootstrap': [False],\
    'max_depth': [185, 190, 195],\
    'max_features': ['sqrt'],\
    'min_samples_leaf': [1, 2],\
    'min_samples_split': [4, 5],\
    'criterion' :['gini']\
}
```

Table 5: Reported Classification Metrics for Random Forest on Non-SMOTE Data

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	1.000e+00	0.9995	0.9999	1.0000	0.9997	0.9996	1	0.9989	0.9986	1	0.9986	1.0000	0.9997	0.9996	0.9997
recall	9.997e-01	0.9999	0.9996	0.9996	0.9996	1.0000	1	0.9987	0.9988	1	1.0000	0.9993	0.9997	0.9996	0.9997
f1-score	9.999e-01	0.9997	0.9997	0.9998	0.9997	0.9998	1	0.9988	0.9987	1	0.9993	0.9997	0.9997	0.9996	0.9997
support	1.027e+04	9281.0000	9375.0000	10757.0000	7559.0000	9810.0000	9831	5515.0000	4911.0000	9447	11313.0000	4343.0000	0.9997	102412.0000	102412.0000

```
if evaluate:
  grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid, cv= 3, \
  verbose=1, return_train_score=True)
  grid_search_rf.fit(X_train, y_train)
  with open('./pickle/grid_search_rf.pickle', 'wb') as f:
    pickle.dump(grid_search_rf, f)

with open('./pickle/grid_search_rf.pickle', 'rb') as f:
  grid_search_rf = pickle.load(f)

grid_search_cv_result = pd.DataFrame(grid_search_rf.cv_results_).\
sort_values(by='rank_test_score',ascending=True)
```

```
get_gridsearch_time(grid_search_rf)
```

```
## GridSearchCV Time Elapsed(s): 283.6076319217682
```

```
best_rf = grid_search_rf.best_estimator_
# Generate predictions
predictions = best_rf.predict(X_test)
```

```
from sklearn.metrics import classification_report
report = pd.DataFrame(classification_report(y_test, predictions, output_dict=True))
```

```
knitr::kable(py$report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Random Forest on Non-SMOTE Data") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
def plot_conf_matrix(conf_matrix, class_labels):
  '''
  Plots a confusion matrix

  Parameters
    conf_matrix: ndarray
      The confusion matrix data

    class_labels: list[str]
      The class labels
  '''
  fig, ax = plt.subplots(figsize=(12,8))
  sn.set(font_scale=0.8)
  sn.heatmap(conf_matrix, annot=True, ax=ax, cmap="Blues", fmt="g")
  ax.set_xlabel('Predicted labels', fontsize=20)
  ax.set_xticklabels(class_labels)
  ax.set_yticklabels(class_labels)
  ax.tick_params(labelsize=15)
  ax.set_ylabel('True labels', fontsize=20)
```

```
  ax.set_title('Confusion Matrix', fontsize=25)
  plt.show()
```

```
from sklearn.metrics import confusion_matrix
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
plot_conf_matrix(conf_matrix, report.keys()[:-3])
```

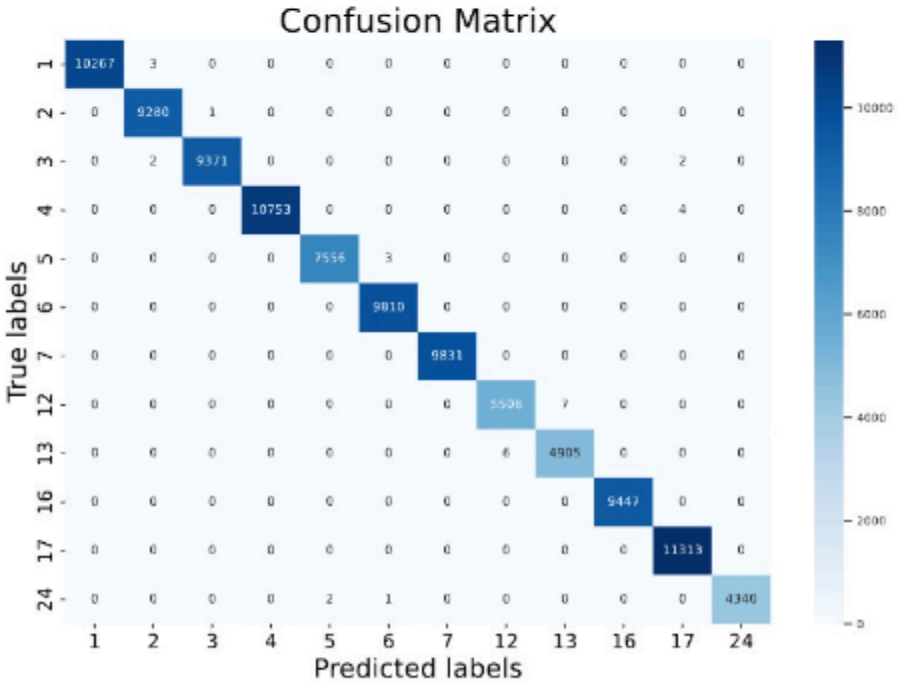


Figure 7: Confusion Matrix for Random Forest on non-SMOTE Data

6.5.2 Random Forest on SMOTE Data

```
if evaluate:
  rand_search_rf_smote = RandomizedSearchCV(estimator = rf, param_distributions = rand_grid,\
  n_iter = 30, cv =3,verbose=2, random_state=441, return_train_score=True)
  rand_search_rf_smote.fit(X_smote, y_smote)
  with open('./pickle/rand_search_rf_smote.pickle', 'wb') as f:
    pickle.dump(rand_search_rf_smote, f)

with open('./pickle/rand_search_rf_smote.pickle', 'rb') as f:
  rand_search_rf_smote = pickle.load(f)

rand_search_smote_cv_result = pd.DataFrame(rand_search_rf_smote.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
plot_cv_results(rand_grid, rand_search_smote_cv_result,\
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=20, width=20)
```

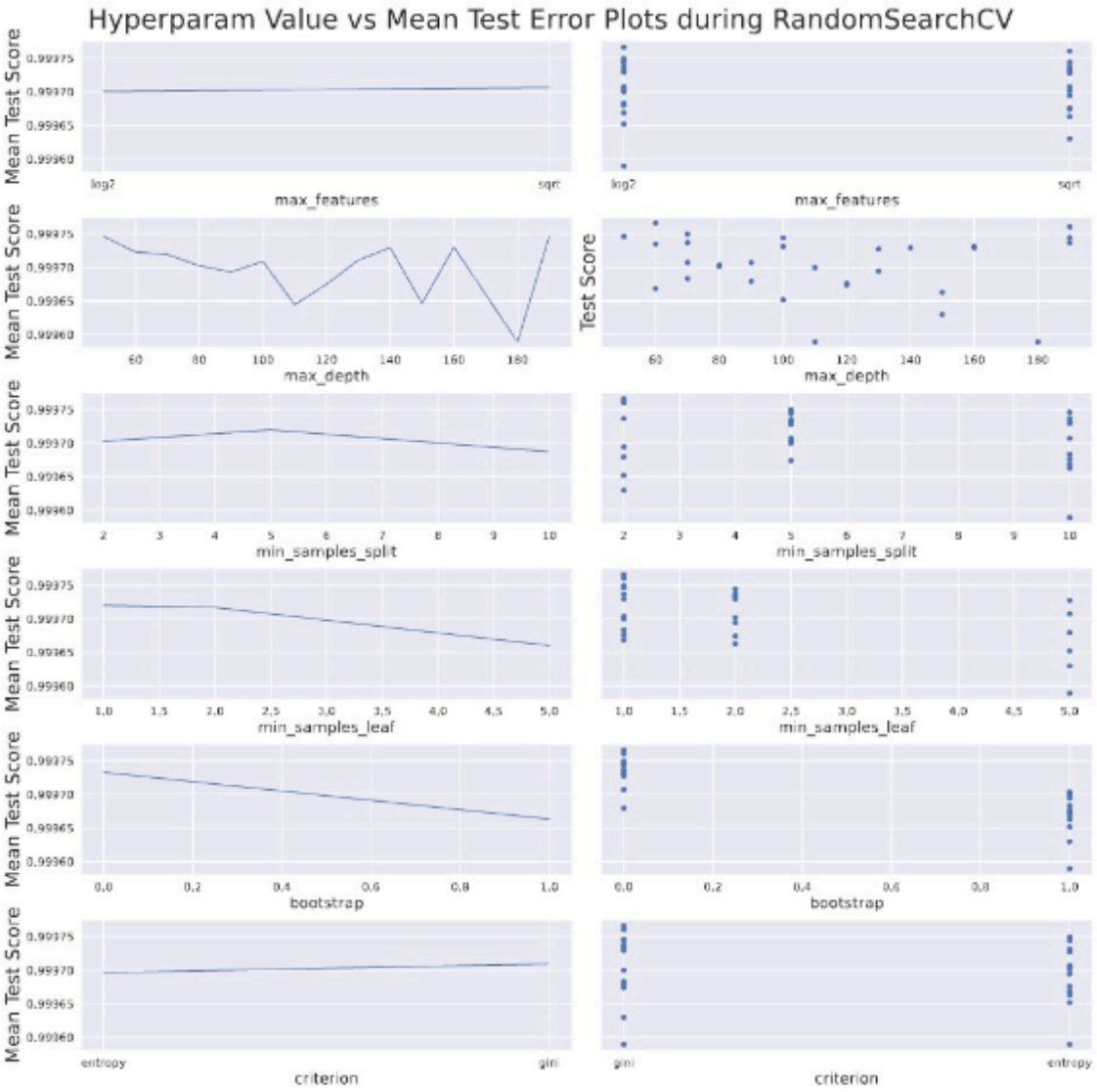


Figure 8: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV on Random Forests for SMOTE Data

```
param_grid = {
    'bootstrap': [False],\
    'max_depth': [185, 190, 195],\
    'max_features': ['sqrt'],\
    'min_samples_leaf': [1, 2],\
    'min_samples_split': [4, 5],\
    'criterion' :['gini']\
}

if evaluate:
```

Table 6: Reported Classification Metrics for Random Forest on SMOTE

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	1.000e+00	0.9992	0.9997	0.9999	0.9995	0.9996	1	0.9987	0.9990	1	0.9995	0.9998	0.9996	0.9996	0.9996
recall	9.996e-01	0.9998	0.9995	0.9996	0.9993	1.0000	1	0.9991	0.9986	1	0.9999	0.9991	0.9996	0.9995	0.9996
f1-score	9.998e-01	0.9995	0.9996	0.9998	0.9995	0.9998	1	0.9989	0.9988	1	0.9997	0.9994	0.9996	0.9996	0.9996
support	1.027e+04	9281.0000	9375.0000	10757.0000	7539.0000	9810.0000	9831	5115.0000	4911.0000	9447	11313.0000	4343.0000	0.9996	102412.0000	102412.0000

```
  grid_search_rf_smote = GridSearchCV(estimator=rf, param_grid=param_grid, \
  cv= 3, verbose=1,return_train_score=True)
  grid_search_rf_smote.fit(X_smote, y_smote)
  with open('./pickle/grid_search_rf_smote.pickle', 'wb') as f:
    pickle.dump(grid_search_rf_smote, f)

with open('./pickle/grid_search_rf_smote.pickle', 'rb') as f:
  grid_search_rf_smote = pickle.load(f)

grid_search_cv_result = pd.DataFrame(grid_search_rf.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
```

```
get_gridsearch_time(grid_search_rf_smote)
```

```
## GridSearchCV Time Elapsed(s): 385.2123155593872
```

```
best_rf = grid_search_rf_smote.best_estimator_
# Generate predictions
predictions = best_rf.predict(X_test)
```

```
from sklearn.metrics import classification_report
report = pd.DataFrame(classification_report(y_test, predictions, output_dict=True))
```

```
knitr::kable(py$report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Random Forest on SMOTE") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
plot_conf_matrix(conf_matrix, report.keys()[:-3])
```

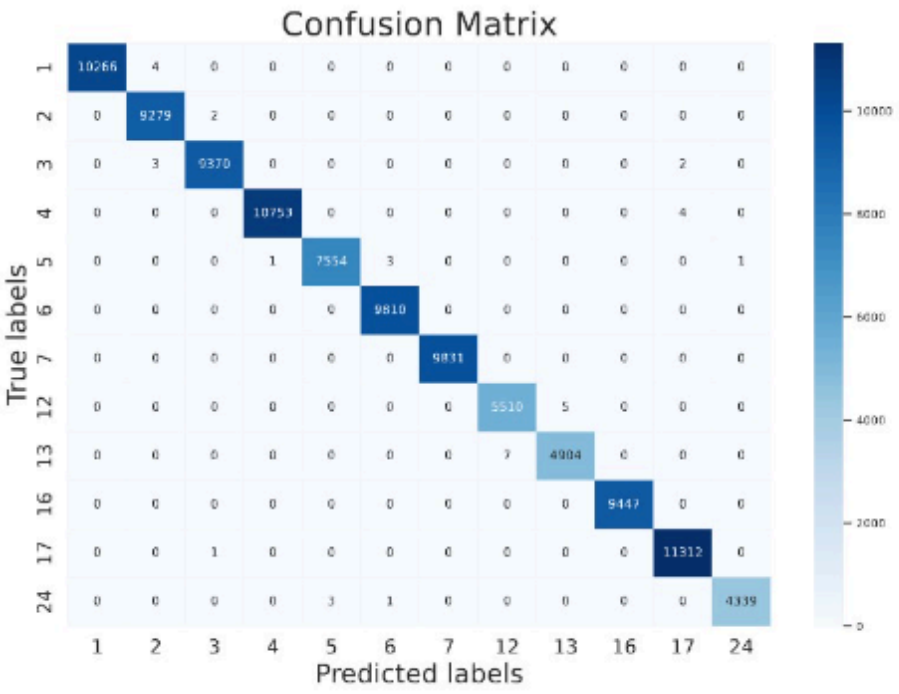
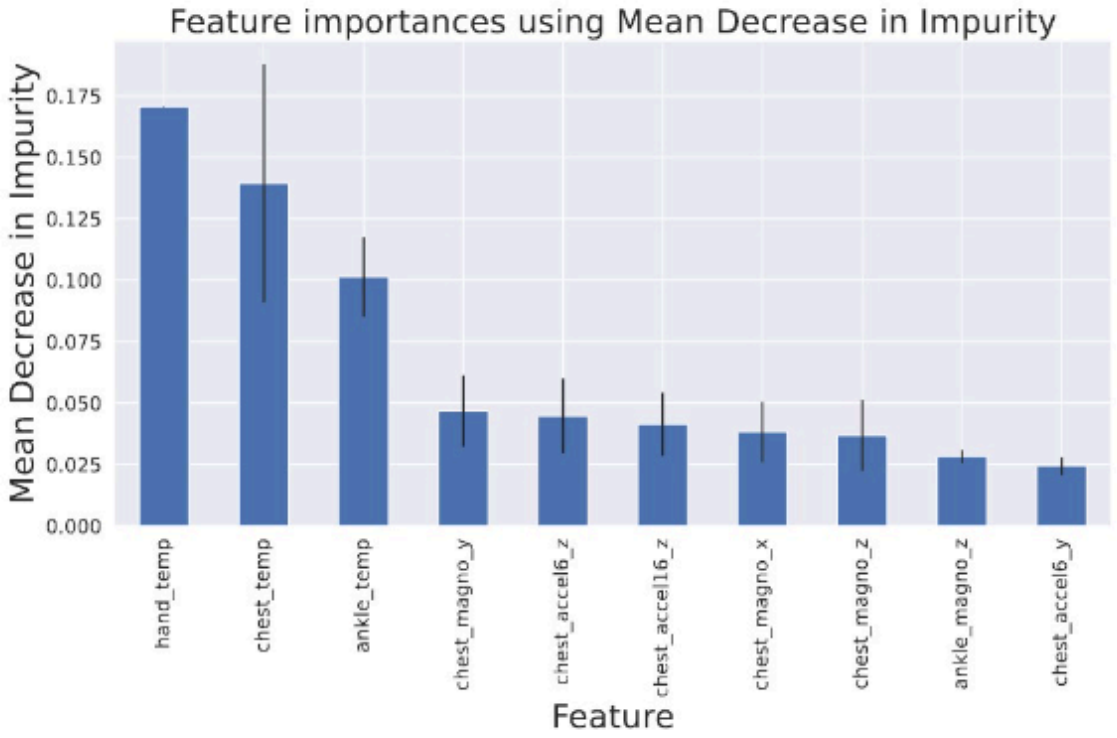
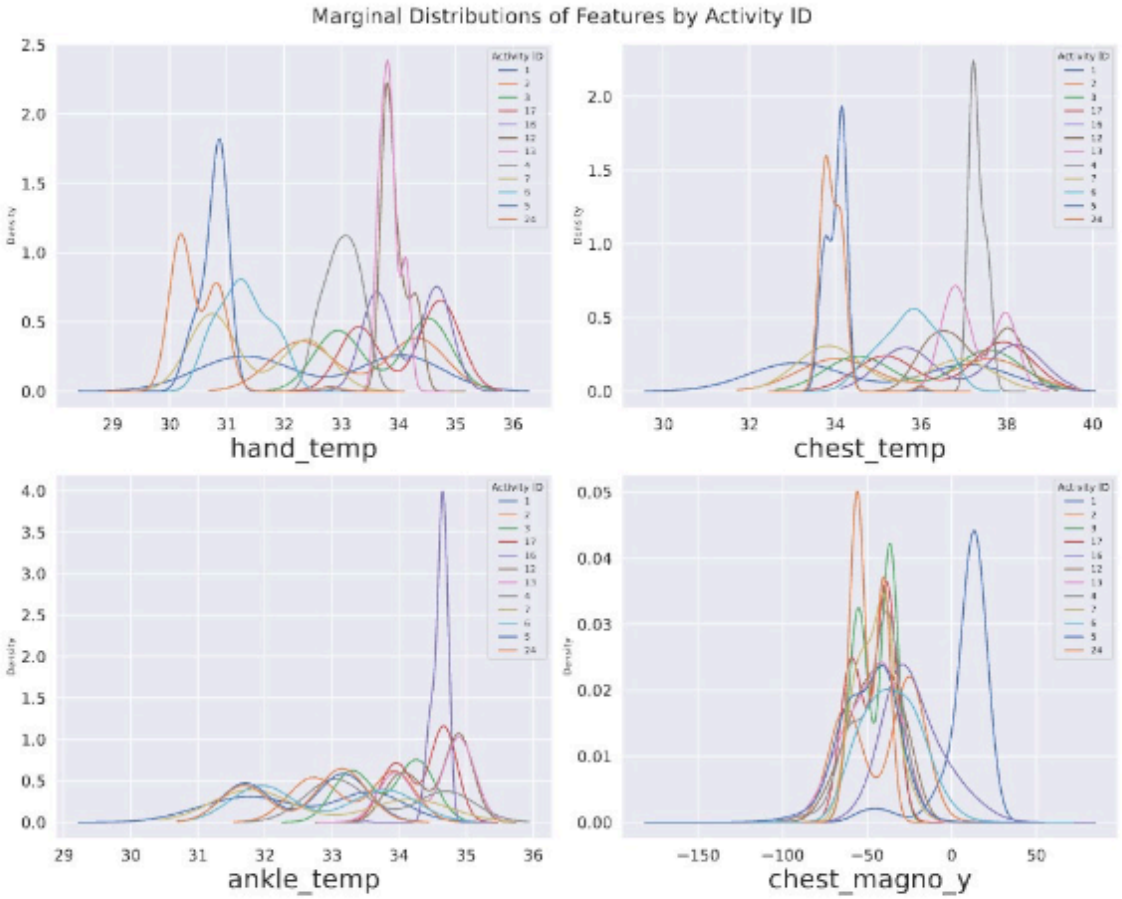


Figure 9: Confusion Matrix for Random Forest on SMOTE Data

```
# Feature importance plot
importances = best_rf.feature_importances_

# Getting top 10 importances
importances = pd.Series(importances, index=X_train.columns).sort_values(ascending=False).head(10)
standard_dev = np.std([tree.feature_importances_ for tree in best_rf.estimators_], axis=0)[:10]
fig, ax = plt.subplots(figsize=(12,8))
importances.plot.bar(yerr=standard_dev, ax=ax)
ax.set_title("Feature importances using Mean Decrease in Impurity", fontsize=25)
ax.set_ylabel("Mean Decrease in Impurity", fontsize=25)
ax.set_xlabel("Feature", fontsize=25)
ax.tick_params(labelsize=15)
fig.tight_layout()
plt.savefig('./plot_images/feature_importance.png', bbox_inches = "tight")
plt.show()
```




```
fig, ax = plt.subplots(figsize=(15,12), nrows=2, ncols=2,sharey=False)
i = -1
for j, col in enumerate(importances.index[:4]):
  if j % 2 == 0:
    i += 1
  for id in data['activity_id'].unique():
    data.loc[data['activity_id'] == id][col].plot.density(ax=ax[i][j%2], bw_method=0.5, label=id)
  ax[i][j%2].set_xlabel(col, fontsize=25)
  ax[i][j%2].legend(title='Activity ID')
  ax[i][j%2].tick_params(labelsize=15)
fig.suptitle("Marginal Distributions of Features by Activity ID", fontsize=20, y=0.99)
fig.tight_layout()
fig.subplots_adjust(top=0.95)
plt.savefig('./plot_images/marginals_by_activity.png')
plt.show()
```



6.6 Stacked Classifier

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler

X_train_base, X_train_meta, y_train_base, y_train_meta = train_test_split(X_train,\
y_train, test_size=0.5, random_state=441, stratify=None)

scaler = StandardScaler()
scaled_X_train_base = scaler.fit_transform(X_train_base)
scaled_X_train_meta = scaler.fit_transform(X_train_meta)
scaled_X_test = scaler.fit_transform(X_test)

X_train_base_smote, X_train_meta_smote, y_train_base_smote, \
y_train_meta_smote = train_test_split(X_smote, y_smote, test_size=0.5, random_state=441, stratify=None)

scaler = StandardScaler()
scaled_X_train_base_smote = scaler.fit_transform(X_train_base_smote)
scaled_X_train_meta_smote = scaler.fit_transform(X_train_meta_smote)
scaled_X_test_smote = scaler.fit_transform(X_test)
```

```
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV

def rand_search_base_classifier(ridge_rand_grid, elasticnet_rand_grid, X_train, y_train, \
ridge_file, elasticnet_file):
    '''
    Performs RandomizedSearchCV for the ridge classifier and elastic net classifier
    based on the parameter grid, ridge_rand_grid and elasticnet_rand_grid, and
    saves the results in the pickle folder

    Parameters:

      ridge_rand_grid: dict
        dictionary of parameter ranges to perform random search over for the
        ridge classifier

      elasticnet_rand_grid: dict
        dictionary of parameter ranges to perform random search over for the
        elasticnet classifier

      X_train: np.array
        a 2D (n_obs x n_features) numpy array of the input features

      y_train: np.array
        a 1D numpy array of length n_features of the input labels

      ridge_file: str
        name of the file to store the random search results for ridge classifier

      elasticnet_file: str
        name of the file to store the random search results for the elasticnet classifier

    Returns:
      None
    '''
    # Random search for Ridge
    ridge_classifier = RidgeClassifier()
    rand_search_ridge = RandomizedSearchCV(estimator = ridge_classifier, \
    param_distributions = ridge_rand_grid, cv = 3, verbose=1, return_train_score=True)
    rand_search_ridge.fit(X_train, y_train)

    # Random search for elasticnet
    elasticnet_classifier = SGDClassifier(loss='log_loss', random_state=1, \
    n_jobs=100, penalty='elasticnet')
    rand_search_elasticnet = RandomizedSearchCV(estimator = elasticnet_classifier\
    , param_distributions = elasticnet_rand_grid, cv = 3, n_iter=30, verbose=1, \
    return_train_score=True)
    rand_search_elasticnet.fit(X_train, y_train)

    # Save random search results
    with open(f"./pickle/{ridge_file}", 'wb') as f:
```

```
        pickle.dump(rand_search_ridge, f)
    with open(f"./pickle/{elasticnet_file}", 'wb') as f:
        pickle.dump(rand_search_elasticnet, f)

def grid_search_base_classifier(ridge_param_grid, elasticnet_param_grid, X_train, y_train, \
ridge_file, elasticnet_file):
    '''
    Performs GridSearchCV for the ridge classifier and elastic net classifier
    based on the parameter grid, ridge_param_grid and elasticnet_param_grid, and
    saves the results in the pickle folder

    Parameters:

      ridge_param_grid: dict
        dictionary of parameter ranges to perform grid search over for the
        ridge classifier

      elasticnet_param_grid: dict
        dictionary of parameter ranges to perform grid search over for the
        elasticnet classifier

      X_train: np.array
        a 2D (n_obs x n_features) numpy array of the input features

      y_train: np.array
        a 1D numpy array of length n_features of the input labels

      ridge_file: str
        name of the file to store the random search results for ridge classifier

      elasticnet_file: str
        name of the file to store the random search results for the elasticnet classifier

    Returns:
      None
    '''
    # Random search for Ridge
    ridge_classifier = RidgeClassifier()
    grid_search_ridge = GridSearchCV(estimator = ridge_classifier, \
    param_grid = ridge_param_grid, cv = 3, verbose=1, return_train_score=True, error_score='raise')
    grid_search_ridge.fit(X_train, y_train)

    # Random search for elasticnet
    elasticnet_classifier = SGDClassifier(loss='log_loss', random_state=1, n_jobs=100, \
    penalty='elasticnet')
    grid_search_elasticnet = GridSearchCV(estimator = elasticnet_classifier, \
    param_grid = elasticnet_param_grid, cv = 3, verbose=1, \
    return_train_score=True, error_score='raise')
    grid_search_elasticnet.fit(X_train, y_train)

    # Save random search results
    with open(f"./pickle/{ridge_file}", 'wb') as f:
        pickle.dump(grid_search_ridge, f)
```

```
    with open(f"./pickle/{elasticnet_file}", 'wb') as f:
        pickle.dump(grid_search_elasticnet, f)
```

```
ridge_rand_grid = {'alpha': [10**x for x in range(-5, 5)]}
elasticnet_rand_grid = {'alpha': [10**x for x in range(-5, 5)],
                        'l1_ratio': np.arange(0, 1, 0.01)}
if evaluate:
  rand_search_base_classifier(ridge_rand_grid, elasticnet_rand_grid, scaled_X_train_base, \
  y_train_base, 'scaled_rand_search_ridge.pickle', 'scaled_rand_search_elasticnet.pickle')

with open('./pickle/scaled_rand_search_ridge.pickle', 'rb') as f:
    scaled_rand_search_ridge = pickle.load(f)
with open('./pickle/scaled_rand_search_elasticnet.pickle', 'rb') as f:
    scaled_rand_search_elasticnet = pickle.load(f)
scaled_rand_search_ridge_cv_result = pd.DataFrame(scaled_rand_search_ridge.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
scaled_rand_search_elasticnet_cv_result = pd.DataFrame(scaled_rand_search_elasticnet.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
```

```
plot_cv_results(ridge_rand_grid, scaled_rand_search_ridge_cv_result,\
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=10, width=20)
```

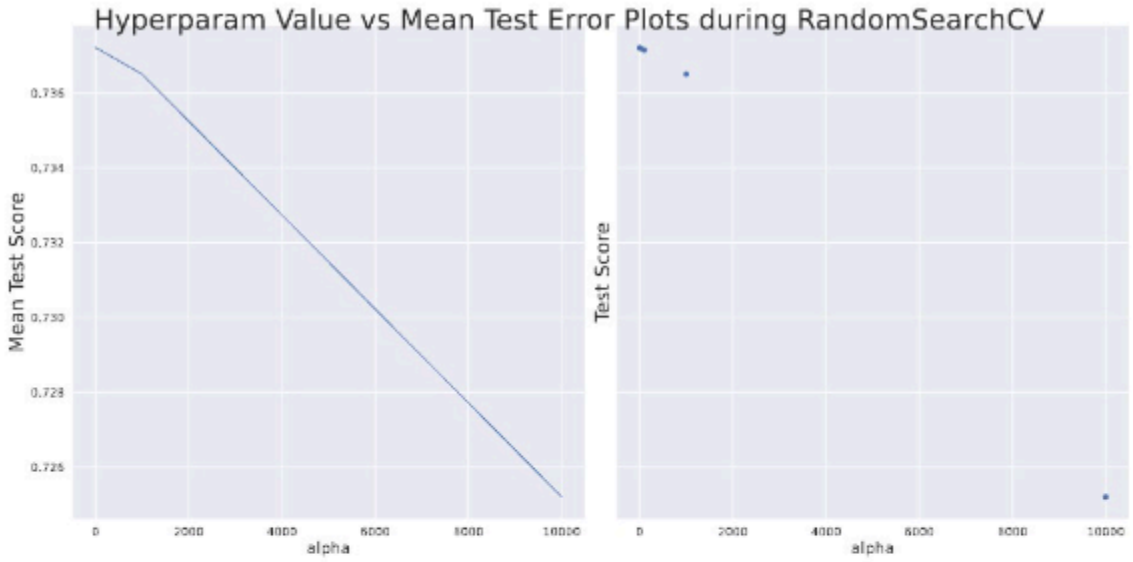


Figure 10: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV for the Ridge base learner on non SMOTE data

```
plot_cv_results(elasticnet_rand_grid, scaled_rand_search_elasticnet_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=20, width=20)
```

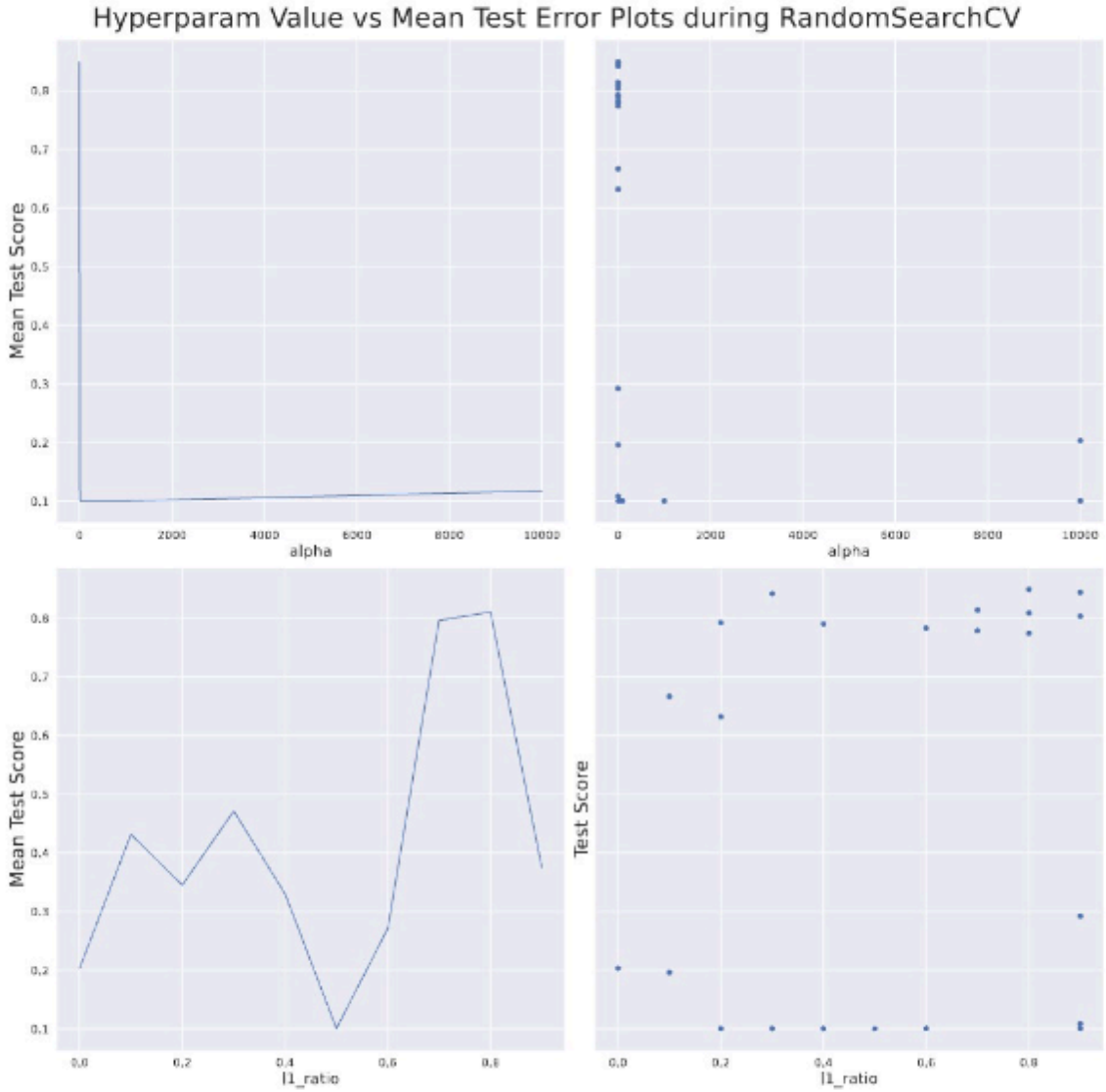


Figure 11: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV for the ElasticNet base learner on non-SMOTE data

```
ridge_param_grid = {'alpha': np.arange(10e-4, 10, 0.1)}
elasticnet_param_grid = {'alpha': np.arange(10e-5, 10e-3, 0.001),
                         'l1_ratio': np.arange(0.7, 0.8, 0.01)}

if evaluate:
  grid_search_base_classifier(ridge_param_grid, elasticnet_param_grid, scaled_X_train_base, \
  y_train_base, 'scaled_grid_search_ridge.pickle', 'scaled_grid_search_elasticnet.pickle')

with open('./pickle/scaled_grid_search_ridge.pickle', 'rb') as f:
```

```
    scaled_grid_search_ridge = pickle.load(f)
with open('./pickle/scaled_grid_search_elasticnet.pickle', 'rb') as f:
    scaled_grid_search_elasticnet = pickle.load(f)

scaled_grid_search_ridge_cv_result = pd.DataFrame(scaled_grid_search_ridge.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
scaled_grid_search_elasticnet_cv_result = pd.DataFrame(scaled_grid_search_elasticnet.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
```

```
from sklearn.ensemble import StackingClassifier

estimators = [\
     ('ridge', scaled_grid_search_ridge.best_estimator_), \
     ('elasticnet', scaled_grid_search_elasticnet.best_estimator_)]

stacked_clf = StackingClassifier(estimators=estimators, \
final_estimator=SGDClassifier(random_state=1, n_jobs=50), n_jobs=-1)

stacked_rand_grid = {
    'final_estimator__alpha': [10**x for x in range(-5, 5)]}

if evaluate:
  scaled_rand_search_stacked = RandomizedSearchCV(estimator = stacked_clf, param_distributions =\
  stacked_rand_grid, cv = 3, verbose=1, return_train_score=True)
  scaled_rand_search_stacked.fit(scaled_X_train_meta, y_train_meta)

  with open('./pickle/scaled_rand_search_stacked.pickle', 'wb') as f:
    pickle.dump(scaled_rand_search_stacked, f)

with open('./pickle/scaled_rand_search_stacked.pickle', 'rb') as f:
  scaled_rand_search_stacked = pickle.load(f)
```

```
scaled_rand_search_stacked_cv_result =\
pd.DataFrame(scaled_rand_search_stacked.cv_results_).sort_values(by='rank_test_score', ascending=True)

plot_cv_results(stacked_rand_grid, scaled_rand_search_stacked_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=10, width=20)
```

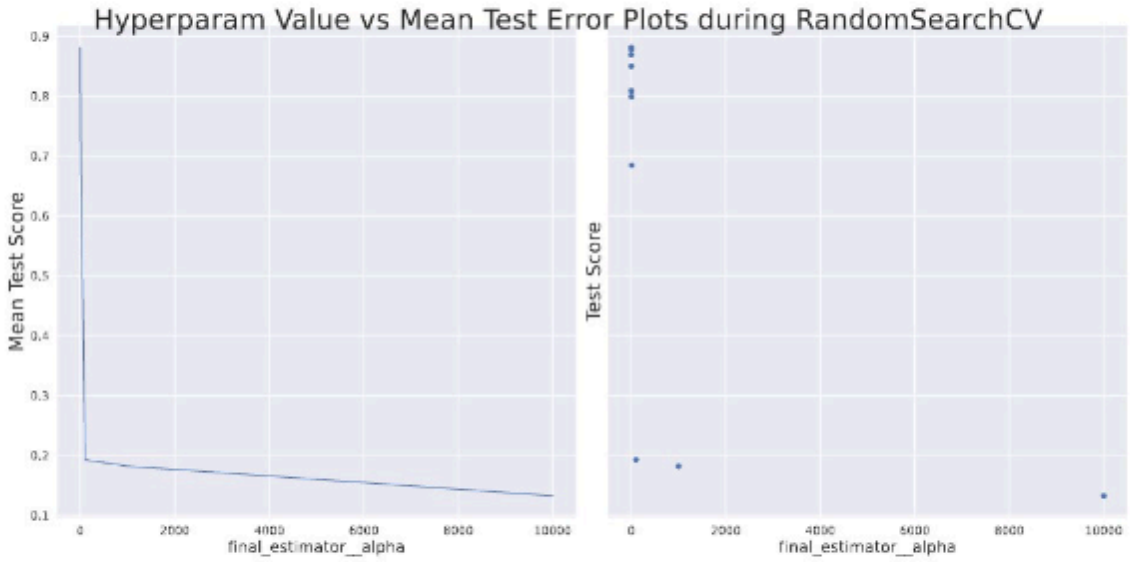


Figure 12: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV for Meta Classifier on SMOTE Data

```
from sklearn.model_selection import GridSearchCV

stacked_param_grid = {
    'final_estimator__alpha': np.arange(1e-5, 1e-3, 0.0001)}

if evaluate:
  scaled_grid_search_stacked = GridSearchCV(estimator = stacked_clf, param_grid = \
  stacked_param_grid, cv = 3, verbose=1, return_train_score=True, error_score='raise')
  scaled_grid_search_stacked.fit(scaled_X_train_meta, y_train_meta)

  with open('./pickle/scaled_grid_search_stacked.pickle', 'wb') as f:
    pickle.dump(scaled_grid_search_stacked, f)

with open('./pickle/scaled_grid_search_stacked.pickle', 'rb') as f:
  scaled_grid_search_stacked = pickle.load(f)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
best_rf = scaled_grid_search_stacked.best_estimator_
# Generate predictions
predictions = best_rf.predict(scaled_X_test)
```

```
from sklearn.metrics import classification_report
report = pd.DataFrame(classification_report(y_test, predictions, output_dict=True))
```

```
knitr::kable(py$report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Stacked Classifier on non-SMOTE Data") %>%
```

Table 7: Reported Classification Metrics for Stacked Classifier on non-SMOTE Data

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.807e-01	0.9502	0.9372	0.7098	0.8131	0.9705	0.7262	0.7587	0.6820	0.9236	0.9703	0.7021	0.8754	0.8565	0.8739
recall	9.689e-01	0.9750	0.9818	0.8492	0.8068	0.9684	0.6295	0.6647	0.7172	0.9431	0.9734	0.7868	0.8754	0.8554	0.8754
f1-score	9.748e-01	0.9625	0.9589	0.8075	0.8097	0.9720	0.6744	0.7086	0.6992	0.9332	0.9718	0.7934	0.8754	0.8552	0.8739
support	1.027e+04	9981.0000	9375.0000	10757.0000	7559.0000	9810.0000	9831.0000	5515.0000	4911.0000	9447.0000	11313.0000	4343.0000	0.8754	102412.0000	102412.0000

```
  kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
plot_conf_matrix(conf_matrix, report.keys()[:-3])
```

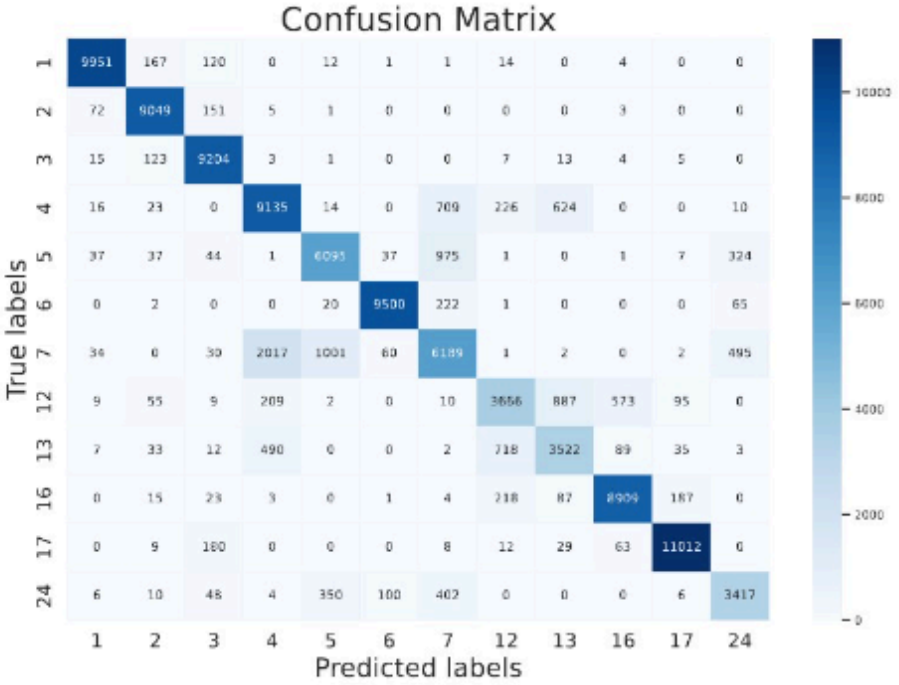


Figure 13: Confusion Matrix for Stacked Classifier on non-SMOTE Data

6.6.0.1 Coarse-to-Fine Tuning on Standard-Scaled, Non-Smote Data

```
ridge_rand_grid = {'alpha': [10**x for x in range(-5, 5)]}
elasticnet_rand_grid = {'alpha': [10**x for x in range(-5, 5)],
                        'l1_ratio': np.arange(0, 1, 0.01)}
if evaluate:
  rand_search_base_classifier(ridge_rand_grid, elasticnet_rand_grid, scaled_X_train_base_smote,\
  y_train_base_smote, 'scaled_rand_search_ridge_smote.pickle',\
  'scaled_rand_search_elasticnet_smote.pickle')
```

```
with open('./pickle/scaled_rand_search_ridge_smote.pickle', 'rb') as f:
    scaled_rand_search_ridge_smote = pickle.load(f)
with open('./pickle/scaled_rand_search_elasticnet_smote.pickle', 'rb') as f:
    scaled_rand_search_elasticnet_smote = pickle.load(f)

scaled_rand_search_ridge_smote_cv_result = pd.DataFrame(scaled_rand_search_ridge_smote.cv_results_)\
.sort_values(by='rank_test_score', ascending=True)
scaled_rand_search_elasticnet_smote_cv_result = pd.DataFrame(\
scaled_rand_search_elasticnet_smote.cv_results_).sort_values(by='rank_test_score', ascending=True)
```

```
plot_cv_results(ridge_rand_grid, scaled_rand_search_ridge_smote_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=10, width=20)
```

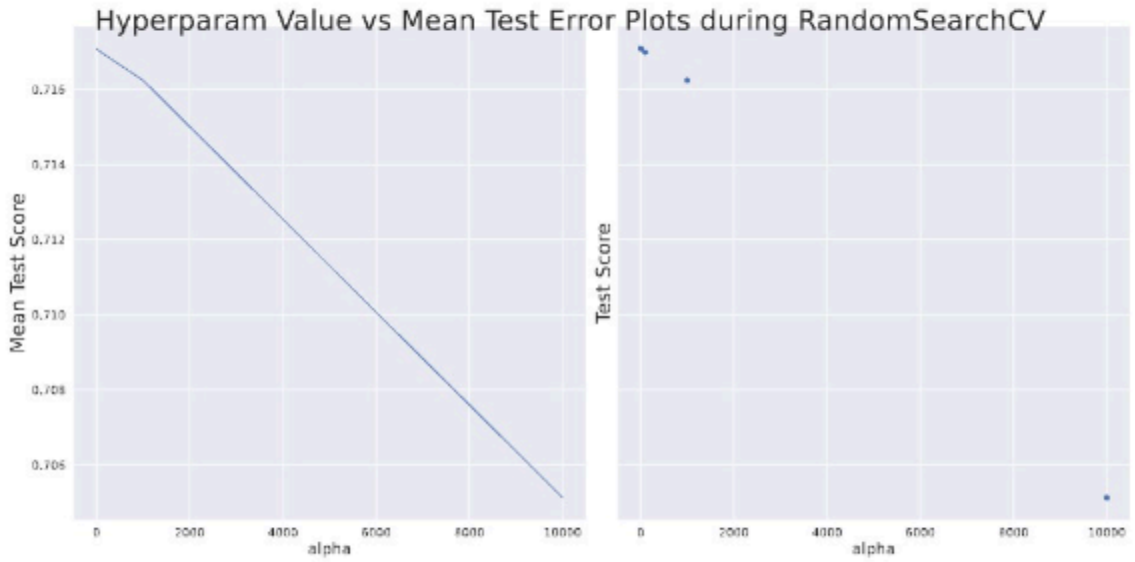


Figure 14: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV on Ridge base learner for SMOTE data

```
plot_cv_results(elasticnet_rand_grid, scaled_rand_search_elasticnet_smote_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=20, width=20)
```

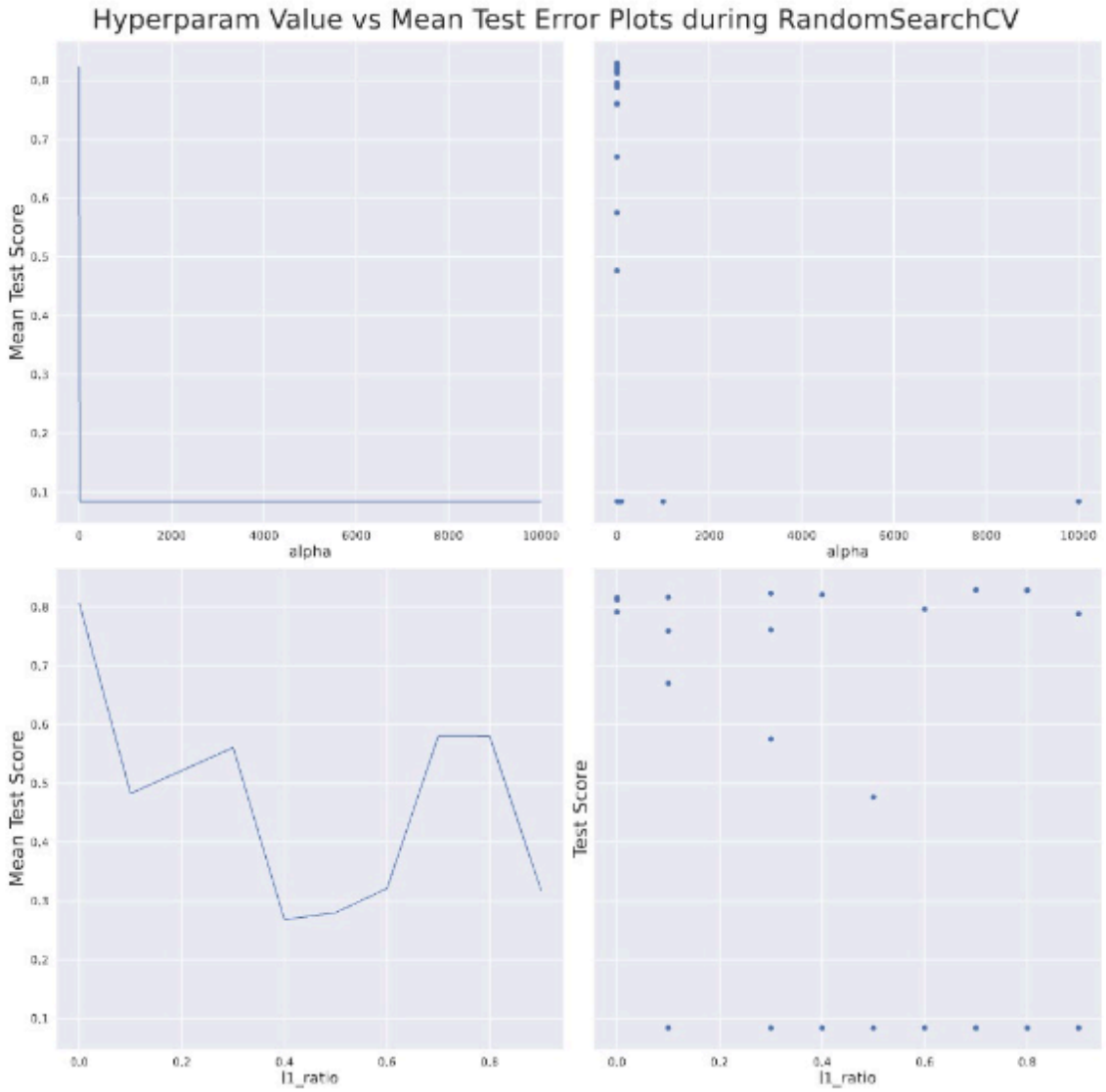


Figure 15: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV on ElasticNet base learner for SMOTE data

```
ridge_param_grid = {'alpha': np.arange(10e-5, 1, 0.01)}
elasticnet_param_grid = {'alpha': np.arange(1e-5, 1e-3, 0.0001),
                         'l1_ratio': np.arange(0.7, 0.8, 0.01)}
if evaluate:
  grid_search_base_classifier(ridge_param_grid, elasticnet_param_grid, \
  scaled_X_train_base_smote, y_train_base_smote, 'scaled_grid_search_ridge_smote.pickle',\
  'scaled_grid_search_elasticnet_smote.pickle')

with open('./pickle/scaled_grid_search_ridge_smote.pickle', 'rb') as f:
```

```
    scaled_grid_search_ridge_smote = pickle.load(f)
with open('./pickle/scaled_grid_search_elasticnet_smote.pickle', 'rb') as f:
    scaled_grid_search_elasticnet_smote = pickle.load(f)
```

```
estimators = [
     ('ridge', scaled_grid_search_ridge_smote.best_estimator_),
     ('elasticnet', scaled_grid_search_elasticnet_smote.best_estimator_)]

stacked_clf = StackingClassifier(
     estimators=estimators, final_estimator=SGDClassifier(random_state=1, n_jobs=50), n_jobs=-1)


stacked_rand_grid = {
    'final_estimator__alpha': [10**x for x in range(-5, 5)]}

if evaluate:
  scaled_rand_search_stacked_smote = RandomizedSearchCV(estimator = stacked_clf, \
  param_distributions = stacked_rand_grid, cv = 3, verbose=1, return_train_score=True)
  scaled_rand_search_stacked_smote.fit(scaled_X_train_meta_smote, y_train_meta_smote)

  with open('./pickle/scaled_rand_search_stacked_smote.pickle', 'wb') as f:
    pickle.dump(scaled_rand_search_stacked_smote, f)

with open('./pickle/scaled_rand_search_stacked_smote.pickle', 'rb') as f:
  scaled_rand_search_stacked_smote = pickle.load(f)

scaled_rand_search_stacked_smote_cv_result = pd.DataFrame(\
scaled_rand_search_stacked_smote.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
```

```
plot_cv_results(stacked_rand_grid, scaled_rand_search_stacked_smote_cv_result, \
title='Hyperparam Value vs Mean Test Error Plots during RandomSearchCV', height=10, width=20)
```

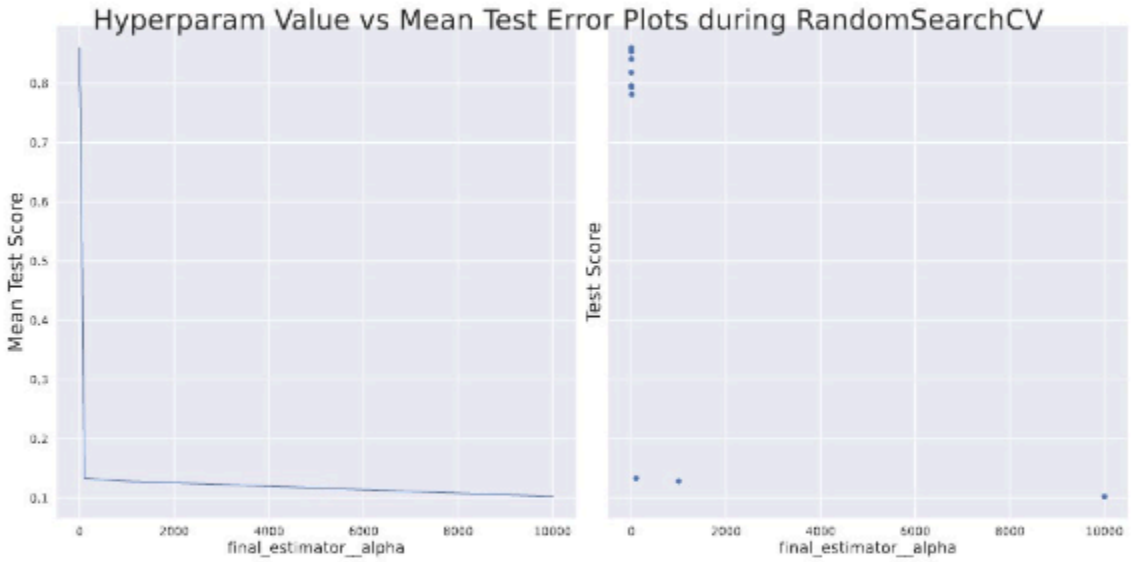


Figure 16: Hyperparam Value vs Mean Test Error Plots during RandomSearchCV on meta learner for SMOTE data

```
stacked_param_grid = {
    'final_estimator__alpha': np.arange(10e-5, 10e-3, 0.001)}

if evaluate:
  scaled_grid_search_stacked_smote = GridSearchCV(estimator = stacked_clf, param_grid = \
  stacked_param_grid, cv = 3, verbose=1, return_train_score=True, error_score='raise')
  scaled_grid_search_stacked_smote.fit(scaled_X_train_meta_smote, y_train_meta_smote)

  with open('./pickle/scaled_grid_search_stacked_smote.pickle', 'wb') as f:
    pickle.dump(scaled_grid_search_stacked_smote, f)

with open('./pickle/scaled_grid_search_stacked_smote.pickle', 'rb') as f:
  scaled_grid_search_stacked_smote = pickle.load(f)
```

```
best_rf = scaled_grid_search_stacked_smote.best_estimator_
# Generate predictions
predictions = best_rf.predict(scaled_X_test_smote)
```

```
from sklearn.metrics import classification_report
report = pd.DataFrame(classification_report(y_test, predictions, output_dict=True))
```

```
knitr::kable(py$report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Stacked Classifier on SMOTE Data") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

Table 8: Reported Classification Metrics for Stacked Classifier on SMOTE Data

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.947e-01	0.9281	0.9440	0.8052	0.7457	0.9693	0.7400	0.6628	0.6485	0.9513	0.9699	0.5277	0.8478	0.8240	0.8579
recall	9.212e-01	0.9558	0.9393	0.7571	0.7066	0.9777	0.5251	0.7630	0.7732	0.9310	0.9563	0.8863	0.8478	0.8409	0.8478
f1-score	9.566e-01	0.9418	0.9420	0.7804	0.7256	0.9735	0.6143	0.7094	0.7054	0.9410	0.9630	0.6615	0.8478	0.8261	0.8484
support	1.027e+04	9281.0000	9375.0000	10757.0000	7559.0000	9810.0000	9831.0000	5535.0000	4911.0000	9447.0000	11313.0000	4343.0000	0.8478	102612.0000	102612.0000

```
# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, predictions)
plot_conf_matrix(conf_matrix, report.keys()[:-3])
```

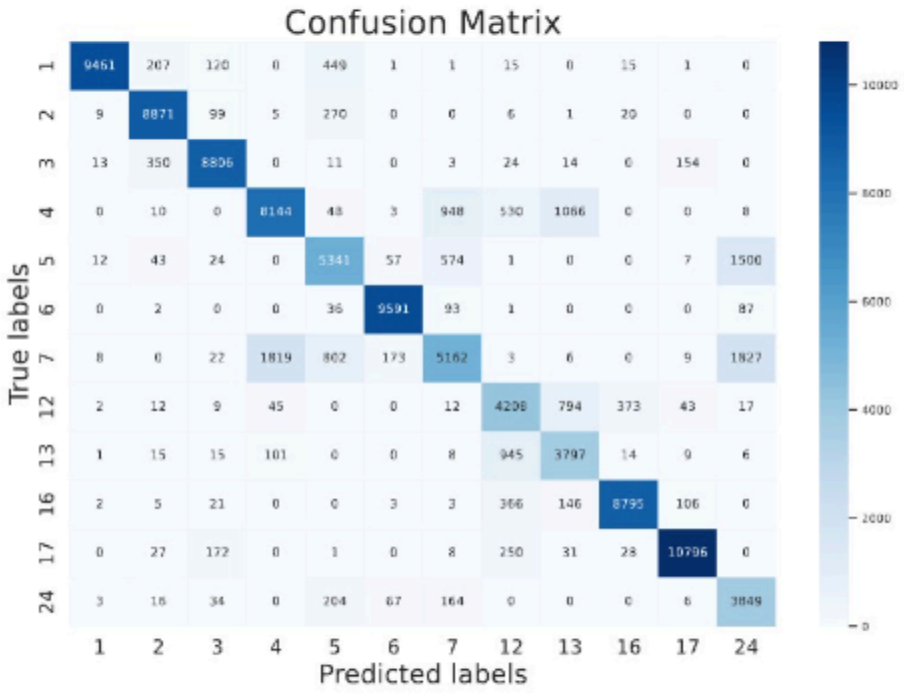


Figure 17: Confusion Matrix for Stacked Classifier on SMOTE data

6.6.0.2 Coarse-to-Fine Tuning on Standard-Scaled, Smote Data

6.6.1 CV tuning base learners and meta classifier combined

```
estimators = [
     ('ridge', RidgeClassifier()),
     ('elasticnet', SGDClassifier(loss='log_loss', random_state=1, n_jobs=50, penalty='elasticnet'))]

clf = StackingClassifier(
     estimators=estimators, final_estimator=SGDClassifier(random_state=1, n_jobs=50), n_jobs=-1)
```

```
scaler = StandardScaler()
scaled_X_train = scaler.fit_transform(X_train)
scaled_X_test = scaler.fit_transform(X_test)
```

```
from numpy import arange
from sklearn.model_selection import RandomizedSearchCV

stack_rand_grid = {'ridge__alpha': [10**x for x in range(-7, 5)],
                   'elasticnet__alpha': [10**x for x in range(-7, 5)],
                   'elasticnet__l1_ratio': np.arange(0, 1, 0.1),
                   'final_estimator__alpha': [10**x for x in range(-7, 5)],
                   'final_estimator__l1_ratio': np.arange(0, 1, 0.1)
                    }

if evaluate:
  rand_search_stack = RandomizedSearchCV(estimator = clf,
                                         param_distributions = stack_rand_grid,
                                         cv = 3, n_iter=10, verbose=2,
                                         return_train_score=True)
  rand_search_stack.fit(scaled_X_train_smote, y_smote)

  with open('./pickle/rand_search_stack.pickle', 'wb') as f:
    pickle.dump(rand_search_stack, f)

if not evaluate:
  with open('./pickle/rand_search_stack.pickle', 'rb') as f:
    rand_search_stack = pickle.load(f)
```

```
get_gridsearch_time(rand_search_stack)
```

6.6.1.1 Coarse-to-Fine tuning on hyperparameters as a whole

```
## GridSearchCV Time Elapsed(s): 2448.3357615470886
```

```
# Display cv results ranked by test score ascending
rand_search_stack_cv_result = pd.DataFrame(rand_search_stack.cv_results_).\
sort_values(by='rank_test_score', ascending=True)
rand_search_stack_cv_result
```

```
##     mean_fit_time  std_fit_time  ...  mean_train_score  std_train_score
## 14      15.129702      0.781746  ...          0.895512         0.001766
## 25      14.862892      0.521415  ...          0.895212         0.001869
## 29      14.215561      0.423519  ...          0.894745         0.003200
## 0       14.906430      0.275625  ...          0.894184         0.002790
## 22      15.210178      1.840266  ...          0.892130         0.002027
## 7       16.264066      0.761677  ...          0.891300         0.001404
## 4       14.697363      1.129438  ...          0.889026         0.001162
## 20      53.696305      3.616740  ...          0.887513         0.001658
## 10      28.859856      2.120408  ...          0.886168         0.001944
## 21      55.660479      5.649388  ...          0.886378         0.002577
## 15      16.826792      1.717504  ...          0.885522         0.000533
## 16      69.643355      6.144005  ...          0.871856         0.002518
## 17      72.249466      5.951259  ...          0.869469         0.009565
```

```
## 12       5.818092      0.488048  ...          0.863133         0.000859
## 2        8.385189      0.813980  ...          0.860006         0.001588
## 11       3.663020      0.165669  ...          0.824773         0.000658
## 13       4.083030      0.263740  ...          0.820369         0.000488
## 1        7.157859      0.988549  ...          0.776178         0.003320
## 19       6.950203      0.547236  ...          0.774535         0.004297
## 26       4.964754      0.525978  ...          0.313064         0.019462
## 27       4.442349      0.112637  ...          0.105186         0.004667
## 28       6.889854      0.332215  ...          0.098505         0.004386
## 18       6.615947      0.373362  ...          0.098505         0.004386
## 6        7.033333      0.275727  ...          0.098505         0.004386
## 8        3.543939      0.110184  ...          0.094656         0.004473
## 23       3.155403      0.090774  ...          0.094656         0.004473
## 5        3.494292      0.180812  ...          0.094656         0.004473
## 9        3.405642      0.132504  ...          0.094656         0.004473
## 24       3.184174      0.082571  ...          0.094656         0.004473
## 3        3.595970      0.192634  ...          0.094656         0.004473
##
## [30 rows x 22 columns]
```

6.7 Neural Network

6.7.1 Pre-processing

```
if evaluate:
  data = load_data(file_root=file_root, subject_ids=subject_ids)
```

```
if evaluate:
  with open('./pickle/data.pickle', 'wb') as f:
        pickle.dump(data, f)
  print("Data saved in ./pickle/data.pickle")
```

```
if not evaluate:
  with open('./pickle/data.pickle', 'rb') as f:
      data = pickle.load(f)
  print("Data loaded from ./pickle/data.pickle")
```

```
## Data loaded from ./pickle/data.pickle
```

```
# Check for missing data
data.isnull().sum()
```

```
## time_stamp             0
## activity_id            0
## heart_rate        465358
## hand_temp           2364
## hand_accel16_x      2364
## hand_accel16_y      2364
## hand_accel16_z      2364
## hand_accel6_x       2364
## hand_accel6_y       2364
## hand_accel6_z       2364
## hand_gyro_x         2364
## hand_gyro_y         2364
## hand_gyro_z         2364
## hand_magno_x        2364
```

```
## hand_magno_y         2364
## hand_magno_z         2364
## hand_orient_x        2364
## hand_orient_y        2364
## hand_orient_z        2364
## hand_orient_w        2364
## chest_temp           1172
## chest_accel16_x      1172
## chest_accel16_y      1172
## chest_accel16_z      1172
## chest_accel6_x       1172
## chest_accel6_y       1172
## chest_accel6_z       1172
## chest_gyro_x         1172
## chest_gyro_y         1172
## chest_gyro_z         1172
## chest_magno_x        1172
## chest_magno_y        1172
## chest_magno_z        1172
## chest_orient_x       1172
## chest_orient_y       1172
## chest_orient_z       1172
## chest_orient_w       1172
## ankle_temp           2286
## ankle_accel16_x      2286
## ankle_accel16_y      2286
## ankle_accel16_z      2286
## ankle_accel6_x       2286
## ankle_accel6_y       2286
## ankle_accel6_z       2286
## ankle_gyro_x         2286
## ankle_gyro_y         2286
## ankle_gyro_z         2286
## ankle_magno_x        2286
## ankle_magno_y        2286
## ankle_magno_z        2286
## ankle_orient_x       2286
## ankle_orient_y       2286
## ankle_orient_z       2286
## ankle_orient_w       2286
## id                      0
## dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder

def preprocess_nn(data):
  '''
  Performs preprocessing on given PAMAP2 data to impute missing values with the mean
  and remove activity_id=0. Drops cols with 'orient' since they are invalid
  according to the dataset readme. Encodes the remaining labels from [0,num_classes) using
  sklearn's label encoder for compatibility with pytorch neural network (nn) functions
  Creates indicator colum
  Parameters:
```

```
    data: pd.DataFrame
      a dataframe containing subject data from the PAMAP2 dataset

  Returns:
    pd.DataFrame
      A preprocessed pandas data frame of subject data from the PAMAP2
      dataset

    dict:
      The mapping used to encode activity labels for decoding purposes
      The dict pairs are (original_id, encoded_id)
  '''
  # Drop cols with orient since they are invalid for this dataset
  data = data.drop([col for col in data.columns if 'orient' in col], axis=1)

  for col in data.columns:
    # Impute columns with mean
    data[col] = data[col].fillna(data[col].mean())
  # Remove activities with id==0
  data = data.drop(data[data['activity_id']==0].index)

  # Encode labels from [0, num_classes) for pytorch
  encoder = LabelEncoder()
  encoder.fit(data['activity_id'])
  encoder_map = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))
  data['activity_id'] = encoder.transform(data['activity_id'])

  # Creates an indicator column for each id
  data = pd.get_dummies(data, columns = ['id'])
  return data, encoder_map
```

```
if evaluate:
  data, encoder_map = preprocess_nn(data)
  num_classes = data['activity_id'].max() + 1
  print('Data Processed, it contains: ' + str(num_classes) + ' classes')
```

```
if evaluate:
  with open('./pickle/data_pp_nn.pickle', 'wb') as f:
    pickle.dump(data, f)
  print("Preprocessed Data saved in ./pickle/data_pp_nn.pickle")
  with open('./pickle/data_pp_encoder.pickle', 'wb') as f:
    pickle.dump(encoder_map, f)
  print("Activity Encoding saved in ./pickle/data_pp_encoder.pickle")
```

```
if not evaluate:
  with open('./pickle/data_pp_nn.pickle', 'rb') as f:
    data  = pickle.load(f)
    num_classes = data['activity_id'].max() + 1
  print("Preprocessed data loaded from ./pickle/data_pp_nn.pickle")
  with open('./pickle/data_pp_encoder.pickle', 'rb') as f:
    encoder_map = pickle.load(f)
  print("Activity Encoding loaded from ./pickle/data_pp_encoder.pickle")
```

```
## Preprocessed data loaded from ./pickle/data_pp_nn.pickle
## Activity Encoding loaded from ./pickle/data_pp_encoder.pickle
```

```
# New mapping for activity ids of the form (original_id, encoded_id)
print(encoder_map)
```

```
## {1: 0, 2: 1, 3: 2, 4: 3, 5: 4, 6: 5, 7: 6, 12: 7, 13: 8, 16: 9, 17: 10, 24: 11}
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

def split_data_nn(data, omit_cols=['time_stamp'], test_frac=0.2, include_val=True, smote=False):
  '''
  Split the given timeseries data into a train set, validation set (optionally) and a test set.
  See params for additional features

  Parameters:

    data: pd.DataFrame
      a dataframe containing subject data from the PAMAP2 dataset

    omit_cols: list[str]
      names of columns to omit from the data when making splits

    val_test_frac: float
      the fraction of data to use in the validation and test splits
      ex. 0.1 will create a 80/10/10 train/val/test split

    include_val
      if True returns a validation set of the same size as the test set
      as specified by test_frac

    smote:
      if True, balances classes in the train data using SMOTE

  Returns:
    dict(pd.Dataframe)
      A dict whose values are pandas dataframe, the dict is of the form
      (X: X_train, y_train)
    Optional: dict(pd.Dataframe)
      A dict whose values are pandas dataframe, the dict is of the form
      (X: X_val, y_val)
    dict(pd.Dataframe)
      A dict whose values are pandas dataframe, the dict is of the form
      (X: X_test, y_test)
  '''
  # Drop cols specified in omit_cols
  data = data.drop(omit_cols, axis=1)

  # Split into features (X) and labels (y)
  X = data.drop(['activity_id'], axis=1)
  y = data['activity_id']

  # Splitting data into train and test split according to test_frac
  if include_val:
    X_train, X_test, y_train, y_test = train_test_split(
```

```
      X, y, test_size=test_frac*2, random_state=441,stratify=y)

    # Splitting test data int test and validation sets
    X_val, X_test, y_val, y_test = train_test_split(
      X_test, y_test, test_size=0.5, random_state=441,stratify=y_test)
  else:
    X_train, X_test, y_train, y_test = train_test_split(
      X, y, test_size=test_frac, random_state=441,stratify=y)

  # Applying Standard Scaling

  # Select columns that don't contain ids since we don't want to scale these
  filter_cols = [col for col in X_train if not col.startswith('id_')]

  scaler = StandardScaler()
  scaler.fit(X_train[filter_cols])
  X_train_scaled = X_train.copy()
  X_test_scaled = X_test.copy()
  if include_val:
    X_val_scaled = X_val.copy()

  X_train_scaled[filter_cols] = scaler.transform(X_train[filter_cols])
  X_test_scaled[filter_cols] = scaler.transform(X_test[filter_cols])
  if include_val:
    X_val_scaled[filter_cols] = scaler.transform(X_val[filter_cols])


  # Optional: Apply smote to balance classes
  if smote:
    sm = SMOTE(random_state=441)
    X_train_scaled, y_train = sm.fit_resample(X_train_scaled, y_train)

  train_data = {'X': X_train_scaled, 'y': y_train}
  test_data = {'X': X_test_scaled, 'y': y_test}

  if include_val:
    val_data = {'X': X_val_scaled, 'y': y_val}
    return train_data, val_data, test_data
  else:
    return train_data, test_data
```

```
if evaluate:
  train_data, val_data, test_data = split_data_nn(data, smote=False)
  print("Successfully split data into train, val and test sets")
```

```
if evaluate:
  with open('./pickle/nn_train_val_test_nosmote.pickle', 'wb') as f:
      pickle.dump([train_data, val_data, test_data], f)
  print("Saved train, val and test data to pickle file ./pickle/nn_train_val_test_nosmote.pickle")
```

```
if not evaluate:
  with open('./pickle/nn_train_val_test_nosmote.pickle', 'rb') as f:
      train_data, val_data, test_data = pickle.load(f)
  print("Loaded train, val and test data from pickle file\
```

```
  ./pickle/nn_train_val_test_nosmote.pickle")
```

```
## Loaded train, val and test data from pickle file  ./pickle/nn_train_val_test_nosmote.pickle
```

```
def seed_everything(seed=441):
    '''
    Seed torch variables for replicable experiments
    '''
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
```

```
if evaluate:
  # Use GPU if available.
  use_cuda = torch.cuda.is_available()
  device = torch.device("cuda" if use_cuda else "cpu")
  if use_cuda:
    print('Using GPU for expriments')
  else:
    print('Using CPU for experiments')
else:
  device = torch.device("cpu")
  print('Using CPU for experiments')
```

```
## Using CPU for experiments
```

```
from torch.nn.modules.activation import LeakyReLU
from torch import nn
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

class PAMAP2_Net(nn.Module):
  '''
  Neural Network Class for classifiying PAMAP2 data
  '''

  def __init__(self, num_classes=12, input_size=42, hidden_dim=32, dropout=0.3,\
  use_log_softmax=False):
      '''
      Defines layers according to specified dimensions

      Params:
        num_classes : int
          The number of classes in the classification problem

        input_size:
          The number of features in the input data

        hidden_dim:
          The number of nodes in the hidden layer of the network
      '''
      super().__init__()
```

```
        self.input_size = input_size
        self.output_size = num_classes
        self.hidden_dim = hidden_dim
        self.num_classes = int(num_classes)
        # This is needed because skorch cv requires a softmax output
        # but NLLLoss requires log softmax
        self.use_log_softmax = use_log_softmax

        # Define layers
        self.fc1 = nn.Linear(input_size, hidden_dim)
        self.relu1 = LeakyReLU()
        self.dropout = nn.Dropout(p=dropout)
        self.fc2 = nn.Linear(hidden_dim, self.output_size)
        self.relu2 = LeakyReLU()
        self.log_softmax = nn.LogSoftmax(dim=1)
        self.softmax = nn.Softmax(dim=1)

        # Losses over epochs
        self.train_losses = []
        self.val_losses = []

    def forward(self, x, debug=False):
        '''
        Performs a forward pass through the network using the input
        data, x and returns a softmax distribution of class probabilities

        Params:
          x : torch.Tensor
            A tensor of the input features yielded from a DataLoader iterator

          debug: bool
            if True, prints the output of the network at each layer for debugging
        '''
        if debug: print('input', x)
        out = self.fc1(x)
        if debug: print('fc1',out)
        out = self.relu1(out)
        if debug: print('relu1',out)
        out = self.dropout(out)
        if debug: print('dropout', out)
        out = self.fc2(out).view(-1, self.output_size)
        if debug: print('fc2',out)
        out = self.relu2(out)
        if debug: print('relu2',out)
        if self.use_log_softmax:
          out = self.log_softmax(out)
        else:
          out = self.softmax(out)
        if debug: print('softmax',out)
        return out

    def train_model(self, train_loader, loss_fn, optimizer):
      '''
```

```
    Executes logic for training the model for a single epoch

    Params:
      train_loader: torch.utils.data.DataLoader
        A torch DataLoader object containing the training data

      loss_fn: torch.nn.modules.loss._Loss
        A torch loss function

      optimizer: torch.optimizer.Optimizer
        A torch optimizer

    Returns:
      float
        The training loss for a single epoch
    '''
    # Set model in train mode
    self.train()

    total_loss, batch_loss, batch_num = 0,0,0
    start_time = time.time()

    for X, y in train_loader:
      # Send tensors to gpu if available
      X, y = X.to(device), y.to(device)
      # Forward pass
      y_pred = model(X)
      # Calculate loss
      loss = loss_fn(y_pred, y.view(-1))

      # Zero gradients and backprop
      optimizer.zero_grad()
      loss.backward()

      # Update params
      optimizer.step()
      # Accumulate loss for all batches
      total_loss += loss.item()
      # Loss for log_interval batches
      batch_loss += loss.item()
      batch_num += 1
      # Batch logging
      if batch_num > 0 and batch_num % log_interval == 0:
          print("Batch: {}/{} |".format(batch_num, len(train_loader)),
                "Time elapsed/batch(ms): {:.5f} |".format(1000 *\
                (time.time()-start_time)/log_interval),
                "Avg Batch Loss: {:.5f} |".format(batch_loss/log_interval))
          batch_loss = 0
          start_time = time.time()
    epoch_loss = total_loss/len(train_loader)
    self.train_losses.append(epoch_loss)
    return epoch_loss
```

```
def evaluate_model(self, data_loader):
  '''
  Executes logic for evaluating the model on the data provided by the data loader

  Params:
    data_loader: torch.utils.data.DataLoader
      A torch DataLoader object containing the validation data

  Returns:
    float
      The validation loss

    dict
      A dictionary containing sklearn classification report metrics
  '''
  # Disables dropout
  self.eval()
  total_loss = 0
  # To store target and predictions for metrics
  targets, preds = [], []

  with torch.no_grad():
    for X, y in data_loader:

        X, y = X.to(device), y.to(device)
        y_pred = model(X)
        loss = loss_fn(y_pred, y.view(-1))
        targets.extend(y.view(-1).tolist())
        preds.extend(torch.argmax(y_pred, dim=1).tolist())
        total_loss += loss.item()
  metrics = classification_report(targets, preds, output_dict=True,\
  target_names=encoder_map.keys())
  epoch_loss = total_loss/len(data_loader)
  self.val_losses.append(epoch_loss)

  return epoch_loss, metrics

def plot_losses(self):
  '''Plots Train and Validation Loss over Epochs'''
  with torch.no_grad():
    fig, ax = plt.subplots(figsize=(12,8))
    # plt.figure()
    ax.set_ylabel('Loss', fontsize=20)
    ax.set_xlabel('Epochs', fontsize=20);
    ax.set_title('Train and Validation Losses vs Epochs', fontsize=25)
    ax.plot(self.train_losses, label="Train Loss")
    ax.plot(self.val_losses, label="Val Loss")
    ax.legend(loc="upper right")
    plt.show()

def learn(self, epochs, loss_fn, optimizer, train_loader, val_loader, save_path):
  '''
  Executes logic for training and validating the model for several epochs and
```

```
    saves the model with the lowest validation loss

    Params:
      train_loader: torch.utils.data.DataLoader
        A torch DataLoader object containing the training data

      val_loader: torch.utils.data.DataLoader
        A torch DataLoader object containing the validation data

      loss_fn: torch.nn.modules.loss._Loss
        A torch loss function

      optimizer: torch.optimizer.Optimizer
        A torch optimizer

      save_path: str
        The path to save the best model to
    '''
    min_val_loss = None
    for epoch in range(epochs):
      start_time = time.time()
      train_loss = self.train_model(train_loader, loss_fn, optimizer)
      mean_val_loss, metrics = self.evaluate_model(val_loader)
      print("Epoch: {}/{} |".format(epoch+1, epochs),
            "Avg Train Loss: {:.5f} |".format(train_loss),
            "Avg Val Loss: {:.5f} |".format(mean_val_loss),
            "Accuracy (Micro): {:.5f} |".format(metrics['accuracy']),
            "Time Elapsed/Epoch(ms): {:.2f} |".format((time.time()-start_time) * 1000))
      if not min_val_loss or mean_val_loss < min_val_loss:
        with open(save_path, 'wb') as f:
            torch.save(self, f)
        if min_val_loss != None:
          print('Validation loss reduced ({:.5f} --> {:.5f}).  Saving model'\
          .format(min_val_loss, mean_val_loss))
        min_val_loss = mean_val_loss
    print("Training/Validation Complete")

  def learn_full(self, epochs, loss_fn, optimizer, train_loader, save_path):
    '''
    Executes logic for training a model on the full training data for several
    epochs

    Params:
      train_loader: torch.utils.data.DataLoader
        A torch DataLoader object containing the training data

      loss_fn: torch.nn.modules.loss._Loss
        A torch loss function

      optimizer: torch.optimizer.Optimizer
        A torch optimizer

      save_path: str
```

```
        The path to save the best model to
    '''
    for epoch in range(epochs):
      start_time = time.time()
      train_loss = self.train_model(train_loader, loss_fn, optimizer)
      print("Epoch: {}/{} |".format(epoch+1, epochs),
            "Avg Train Loss: {:.5f} |".format(train_loss),
            "Time Elapsed/Epoch(ms): {:.2f} |".format((time.time()-start_time) * 1000))
    print("Training on full data complete. Saving model...")
    with open(save_path, 'wb') as f:
            torch.save(self, f)

  def test_model(self, test_loader):
    '''
    Executes logic for testing a model on a test set

    Params:
      test_loader: torch.utils.data.DataLoader
        A torch DataLoader object containing the test data

    Returns:
      dict:
        a dictionary containing classification metrics produced
        by sklearn's classification_report()

      ndarray:
        a confusion matrix
    '''
    # Disables dropout
    model.eval()
    # To store target and predictions for metrics
    targets, preds = [], []

    with torch.no_grad():
      for X, y in test_loader:
          X, y = X.to(device), y.to(device)
          y_pred = model(X)
          targets.extend(y.view(-1).tolist())
          preds.extend(torch.argmax(y_pred, dim=1).tolist())
    metrics = classification_report(targets, preds, output_dict=True, digits=4,\
    target_names=encoder_map.keys())
    conf_matrix = confusion_matrix(targets, preds)
    print("Model Testing Complete...Returning metrics")
    return metrics, conf_matrix
```

```
# Training Hyperparameters
lr= 0.001
batch_size = 32
epochs = 10
log_interval = 100 # Interval at which to print batch progress while training
```

```
from torch.utils.data import DataLoader
from torch.utils.data import TensorDataset
```

```
data_loader_params = {'batch_size': batch_size, 'shuffle': True, 'num_workers': 0}

def generate_data_loader(data_loader_params, data):
  '''
  Creates a torch DataLoader object for provided data

  Params:
    data_loader_params: dict
      a dictionary of parameters and their values to be provided to the torch
      DataLoader object. See pytorch documentation for parameters and their meanings

    data: Optional(dict)
      a dict of the form {X: X_data, y: y_data} where X_data, y_data are pandas DataFrames

  Returns:
    torch.utils.data.DataLoader
      A torch DataLoader object
  '''
  # Set up TensorDatasets
  dataset = TensorDataset(torch.tensor(data['X'].values,dtype=torch.float32),\
  torch.tensor(data['y'].values, dtype=torch.long))

  # Set up DataLoaders
  data_loader = DataLoader(dataset, **data_loader_params)

  return data_loader
```

```
if evaluate:
  seed_everything()
  train_loader = generate_data_loader(data_loader_params, train_data)
  val_loader = generate_data_loader(data_loader_params, val_data)
  test_loader = generate_data_loader(data_loader_params, test_data)
  print("Successfully generated DataLoaders")
```

```
if evaluate:
  with open('./pickle/nn_train_val_test_loaders_nosmote.pickle', 'wb') as f:
      pickle.dump([train_loader, val_loader, test_loader], f)
  print("Saved train, val and test loaders to pickle file\
  ./pickle/nn_train_val_test_loaders_nosmote.pickle")
```

```
if not evaluate:
  with open('./pickle/nn_train_val_test_loaders_nosmote.pickle', 'rb') as f:
      train_loader, val_loader, test_loader = pickle.load(f)
  print("Loaded train and test data from pickle file ./pickle/nn_train_val_test_no_smote.pickle")
```

```
## Loaded train and test data from pickle file ./pickle/nn_train_val_test_no_smote.pickle
```

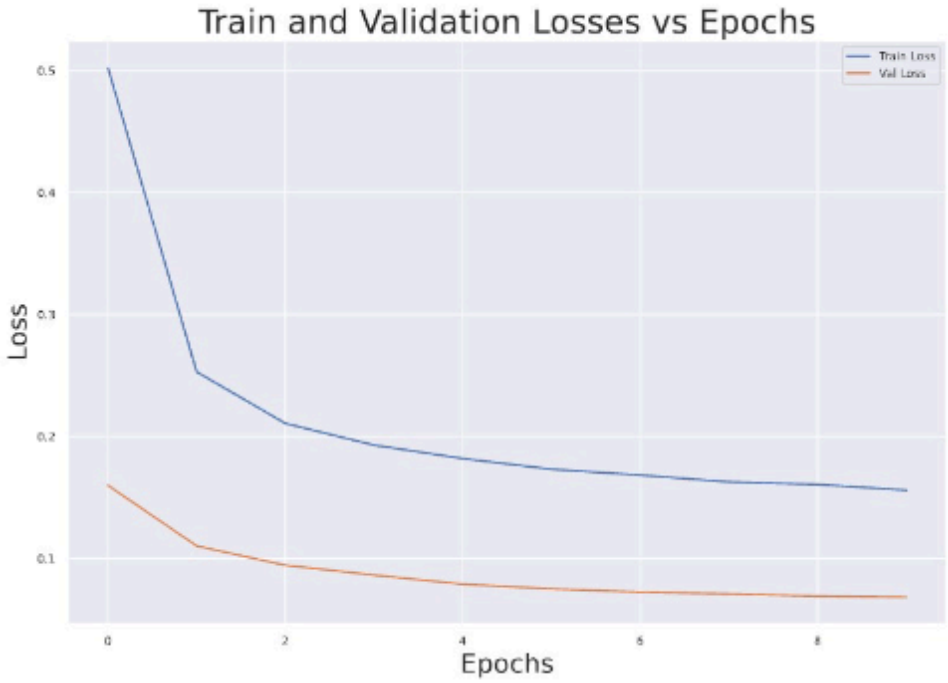
```
if evaluate:
  # Seed before to initialize with same weights
  seed_everything()
  model = PAMAP2_Net(num_classes=num_classes, use_log_softmax=True)
  model.to(device)
  loss_fn = torch.nn.NLLLoss().to(device)
  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
  model.learn(epochs=epochs, loss_fn=loss_fn, optimizer=optimizer, train_loader=train_loader,\
```

```
  val_loader=val_loader, save_path='./pickle/model_nosmote.pt')
```

```
if not evaluate:
  with open('./pickle/model_nosmote.pt', 'rb') as f:
      model = torch.load(f)
  print("Loaded model trained on data without smote from pickle file ./pickle/model_nosmote.pt")
```

```
## Loaded model trained on data without smote from pickle file ./pickle/model_nosmote.pt
```

```
model.plot_losses()
```




```
# Seed before evaluation so we get the same result whether we train or load the model
seed_everything()
metrics, conf_matrix = model.test_model(test_loader=test_loader)
```

```
## Model Testing Complete...Returning metrics
```

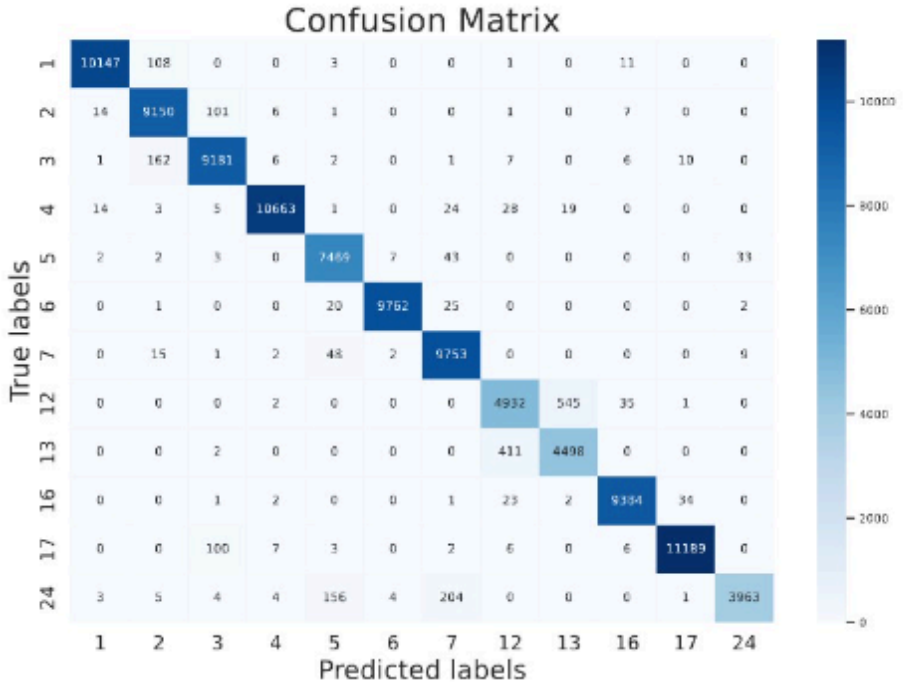
```
metrics_report = pd.DataFrame(metrics)
```

```
knitr::kable(py$metrics_report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Neural Net without SMOTE") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
plot_conf_matrix(conf_matrix, encoder_map.keys())
```

Table 9: Reported Classification Metrics for Neural Net without SMOTE

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.967e-01	0.9887	0.9769	0.9973	0.9686	0.9987	0.9702	0.9118	0.8882	0.9931	0.9659	0.9890	0.9773	0.9713	0.9775
recall	9.880e-01	0.9860	0.9792	0.9913	0.9881	0.9951	0.9922	0.8943	0.9159	0.9903	0.9800	0.9123	0.9773	0.9687	0.9773
f1-score	9.923e-01	0.9873	0.9781	0.9943	0.9783	0.9969	0.9810	0.9030	0.9019	0.9932	0.9725	0.9491	0.9773	0.9699	0.9773
support	1.027e+04	9280.0000	9176.0000	10757.0000	7559.0000	9810.0000	9830.0000	5515.0000	4911.0000	9447.0000	11313.0000	4344.0000	0.9773	102412.0000	102412.0000



6.7.2 Neural Net with SMOTE

```
if evaluate:
  train_data, val_data, test_data = split_data_nn(data, smote=True)
  print("Successfully split data into train, val and test sets with SMOTE")
```

```
if evaluate:
  with open('./pickle/nn_train_val_test_smote.pickle', 'wb') as f:
      pickle.dump([train_data, val_data, test_data], f)
  print("Saved train, val and test data to pickle file ./pickle/nn_train_val_test_smote.pickle")
```

```
if not evaluate:
  with open('./pickle/nn_train_val_test_smote.pickle', 'rb') as f:
      train_data, val_data, test_data = pickle.load(f)
  print("Loaded train, val and test data from pickle file\
  ./pickle/nn_train_val_test_smote.pickle")
```

```
## Loaded train, val and test data from pickle file  ./pickle/nn_train_val_test_smote.pickle
```

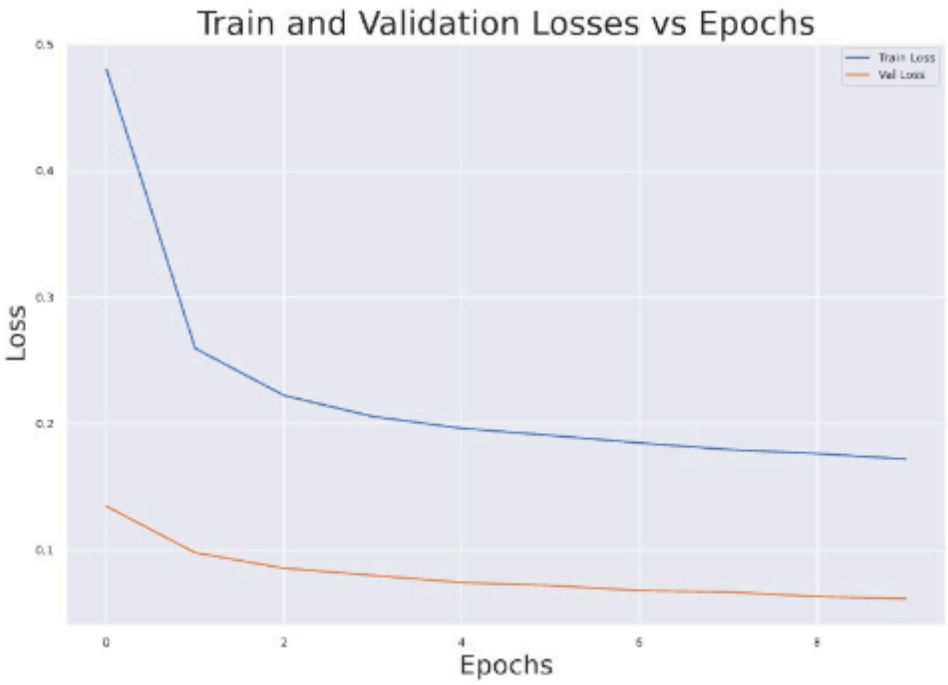
```
if evaluate:
  seed_everything()
  train_loader = generate_data_loader(data_loader_params, train_data)
  val_loader = generate_data_loader(data_loader_params, val_data)
  test_loader = generate_data_loader(data_loader_params, test_data)
  print("Successfully generated DataLoaders with SMOTE data")
```

```
if evaluate:
  # Seed before to initialize with same weights
  seed_everything()
  model = PAMAP2_Net(num_classes=num_classes, use_log_softmax=True)
  model.to(device)
  loss_fn = torch.nn.NLLLoss().to(device)
  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
  model.learn(epochs=epochs, loss_fn=loss_fn, optimizer=optimizer, train_loader=train_loader,\
  val_loader=val_loader, save_path='./pickle/model_smote.pt')
```

```
if not evaluate:
  with open('./pickle/model_smote.pt', 'rb') as f:
      model = torch.load(f)
  print("Loaded model trained on data without smote from pickle file ./pickle/model_smote.pt")
```

```
## Loaded model trained on data without smote from pickle file ./pickle/model_smote.pt
```

```
model.plot_losses()
```




```
# Seed before evaluation so we get the same result whether we train or load the model
seed_everything()
```

Table 10: Reported Classification Metrics for Neural Net Smote

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.958e-01	0.9882	0.9712	0.9965	0.9750	0.9984	0.9904	0.9401	0.9155	0.9963	0.9932	0.9586	0.9818	0.9782	0.9819
recall	9.929e-01	0.9785	0.9967	0.9896	0.9724	0.9956	0.9909	0.9300	0.9444	0.9892	0.9868	0.9583	0.9818	0.9770	0.9818
f1-score	9.943e-01	0.9833	0.9838	0.9931	0.9760	0.9970	0.9856	0.9353	0.9297	0.9927	0.9900	0.9584	0.9818	0.9786	0.9818
support	1.027e+04	9280.0000	9376.0000	10757.0000	7559.0000	9810.0000	9830.0000	5515.0000	4911.0000	9447.0000	11313.0000	4364.0000	0.9818	102412.0000	102412.0000

```
metrics, conf_matrix = model.test_model(test_loader=test_loader)
```

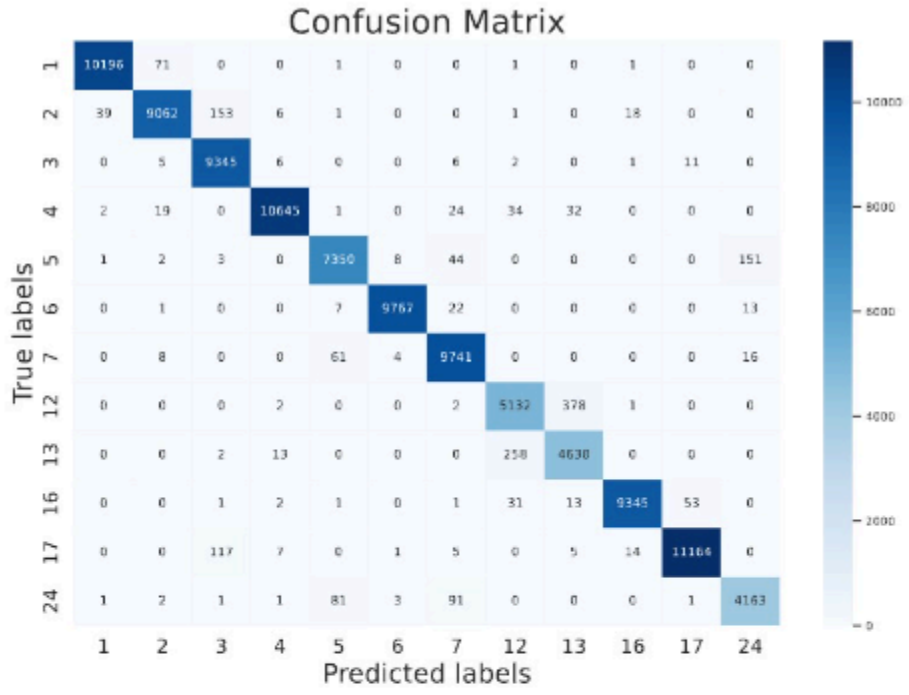
```
## Model Testing Complete...Returning metrics
```

```
metrics_report = pd.DataFrame(metrics)
```

```
knitr::kable(py$metrics_report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Neural Net Smote") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
plot_conf_matrix(conf_matrix, encoder_map.keys())
```

```
## <string>:12: RuntimeWarning: More than 20 figures have been opened. Figures created through the pypl
```



6.7.3 GridSearch CV for Tuning

```
# Merge train and validation sets
train_data['X'] = pd.concat([train_data['X'], val_data['X']], axis=0)
```

```
train_data['y'] = pd.concat([train_data['y'], val_data['y']], axis=0)
```

```
from skorch import NeuralNetClassifier
from skorch.dataset import Dataset
from sklearn.model_selection import GridSearchCV

net = NeuralNetClassifier(PAMAP2_Net,
                          module__num_classes=num_classes,
                          module__use_log_softmax=False,
                          max_epochs=10,
                          lr=0.001,
                          batch_size=32,
                          verbose=1,
                          optimizer=torch.optim.Adam,
                          device=device,
                          train_split=False)
param_grid = {
    'module__dropout':[0, 0.3],
    'batch_size': [16,32,64]
}
```

```
if evaluate:
  seed_everything()
  nn_grid_search = GridSearchCV(net, param_grid, refit=False, cv=3, scoring='accuracy', verbose=2)
  nn_grid_search.fit(torch.tensor(train_data['X'].values,dtype=torch.float32),\
       torch.tensor(train_data['y'], dtype=torch.long))
```

```
if evaluate:
  with open('./pickle/nn_grid_search.pickle', 'wb') as f:
      pickle.dump(nn_grid_search, f)
  print("Saved NN Grid Search Result to pickle file ./pickle/nn_grid_search.pickle")
```

```
if not evaluate:
  with open('./pickle/nn_grid_search.pickle', 'rb') as f:
      nn_grid_search = pickle.load(f)
  print("Loaded NN Grid Search Result from pickle file ./pickle/nn_grid_search.pickle")
```

```
## Loaded NN Grid Search Result from pickle file ./pickle/nn_grid_search.pickle
##
## /srv/jupyter_python3-extra/lib/python3.10/site-packages/sklearn/base.py:329: UserWarning: Trying to
## https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
##   warnings.warn(
```

```
get_gridsearch_time(nn_grid_search)
```

```
## GridSearchCV Time Elapsed(s): 4349.484916687012
```

```
print("best score: {:.3f}, best params: {}".format(nn_grid_search.best_score_,\
nn_grid_search.best_params_))
```

```
## best score: 0.894, best params: {'batch_size': 16, 'module__dropout': 0}
```

6.7.4 Fitting Final Model

```
if evaluate:
  # Seed before data loader for consistency
```

Table 11: Reported Classification Metrics for Final Neural Net

	1	2	3	4	5	6	7	12	13	16	17	24	accuracy	macro avg	weighted avg
precision	9.990e-01	0.9895	0.9852	0.9950	0.9954	0.9994	0.9930	0.9701	0.9783	0.9988	0.9991	0.9874	0.9926	0.9929	0.9926
recall	9.946e-01	0.9939	0.9883	0.9972	0.9919	0.9987	0.9964	0.9811	0.9956	0.9969	0.9905	0.9924	0.9926	0.9911	0.9926
f1-score	9.970e-01	0.9917	0.9867	0.9961	0.9936	0.9990	0.9947	0.9756	0.9719	0.9979	0.9948	0.9899	0.9926	0.9919	0.9926
support	1.027e+04	9280.0000	9176.0000	10757.0000	7559.0000	9810.0000	9830.0000	5515.0000	4911.0000	9647.0000	11313.0000	4364.0000	0.9926	102612.0000	102612.0000

```
  seed_everything()
  data_loader_params = {'batch_size': 16, 'shuffle': True, 'num_workers': 0}
  train_loader = generate_data_loader(data_loader_params, train_data)
  # Seed before model to initialize weights consistently
  seed_everything()
  model = PAMAP2_Net(num_classes=num_classes, use_log_softmax=True)
  model.to(device)
  loss_fn = torch.nn.NLLLoss().to(device)
  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
  model.learn_full(epochs=epochs, loss_fn=loss_fn, optimizer=optimizer, train_loader=train_loader,\
  save_path='./pickle/full_model.pt')
```

```
if not evaluate:
  with open('./pickle/full_model.pt', 'rb') as f:
      model = torch.load(f)
  print("Loaded model trained on full data with smote from pickle file ./pickle/full_model.pt")
```

```
## Loaded model trained on full data with smote from pickle file ./pickle/full_model.pt
```

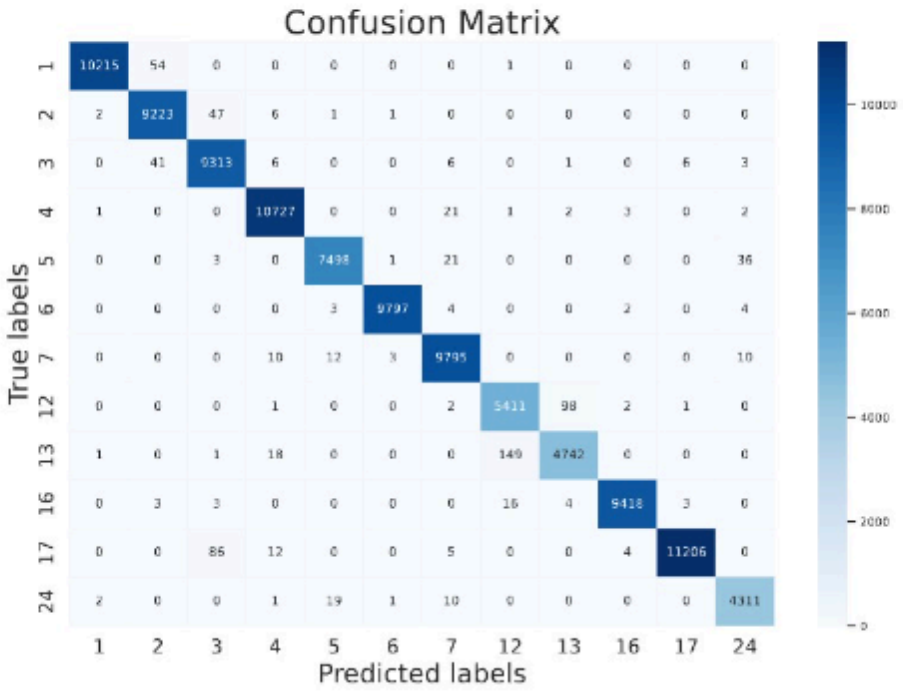
```
seed_everything()
metrics, conf_matrix = model.test_model(test_loader=test_loader)
```

```
## Model Testing Complete...Returning metrics
```

```
metrics_report = pd.DataFrame(metrics)
```

```
knitr::kable(py$metrics_report, booktabs = TRUE, digits=4,
             caption="Reported Classification Metrics for Final Neural Net") %>%
kable_styling(latex_options = c("striped", "scale_down"), full_width = F)
```

```
plot_conf_matrix(conf_matrix, encoder_map.keys())
```


Confusion Matrix
True labels
Predicted labels
1
2
3
4
5
6
7
12
13
16
17
24
10000
8000
6000
4000
2000
0