Methodology

Given that the aim of this coursework is to produce models that accurately predict the activity of a smart watch user at a given point of time, the ML algorithms that we propose are ones for classification. There are several algorithms that are capable of this sort of task and this report will look to propose and discuss the results of implementing such algorithms. The first types of algorithms that we used are supervised learning algorithms such as logistic regression, random forests, SVMs and GBMs. To begin with we used random forests, SVMs and GBMs and training them using the metadata and creating predictions to submit to Kaggle to evaluate the accuracy of the model.

From here we began the process of hyperparameter tuning and using the additional extracted features from the signal data. The hyperparameter tuning depended on the complexity of the model as more complex stacked models such as the one implemented in our final solution required the use of a random search due to the number of hyperparameters that we were looking to tune to develop the best model. This was due to the time that it would take to perform a grid search was not feasible and although the results of a grid search would be better than those of a random search due to the exhaustive nature of the hyperparameter space. Grid searches were however suitable for use on the individual random forest and GBM models. We found that our model performed best when adding the test file and using it as validation data when training the model along with the metadata and using the Kaggle file for predicting the user activity. We found this worked best due to the additional data that we were able to use when training the model, along with the additional features that we were able to extract from the signal’s dataset.

Xgboost was another of the ensemble methods we used, we thought that testing different ensemble methods would work well in evaluating which ones would work best in a stacked model. As with all the models that we tried we used hyperparameter tuning this time in the form of a random search to try and optimise the model's accuracy with the extracted features and feature selection, to reduce the complexity of the model.

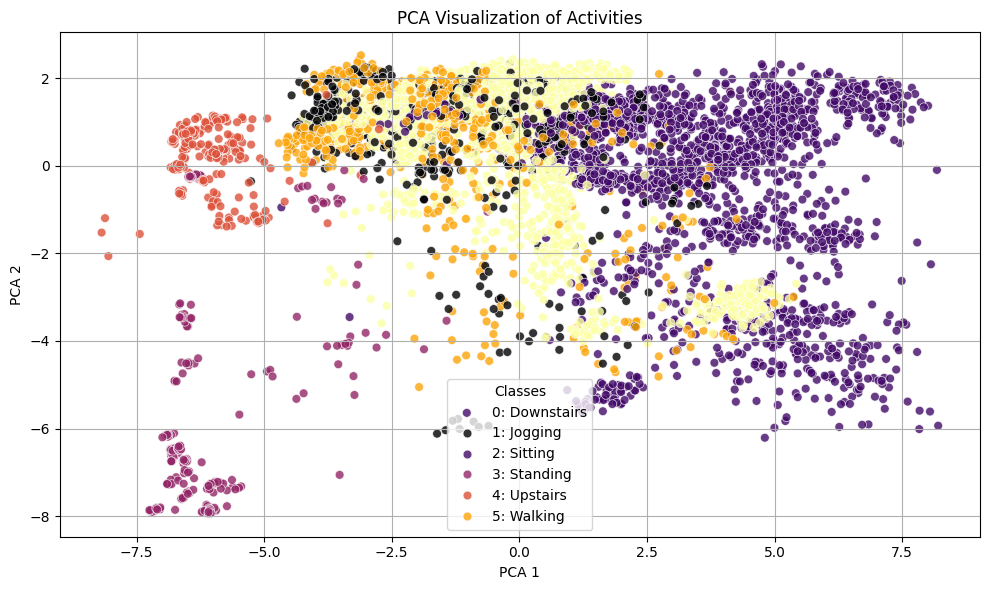
One of the other models that we attempted to produce was an artificial neural network as we wanted to see if a deep learning model like this could outperform our machine learning models. We thought this might work due to the nature of our data and the fact that it is non-linear, ANNs are more capable of learning non-linear features and can perform well on large datasets. Like with the machine learning models we implemented random search hyperparameter tuning to try and optimise the model's accuracy.

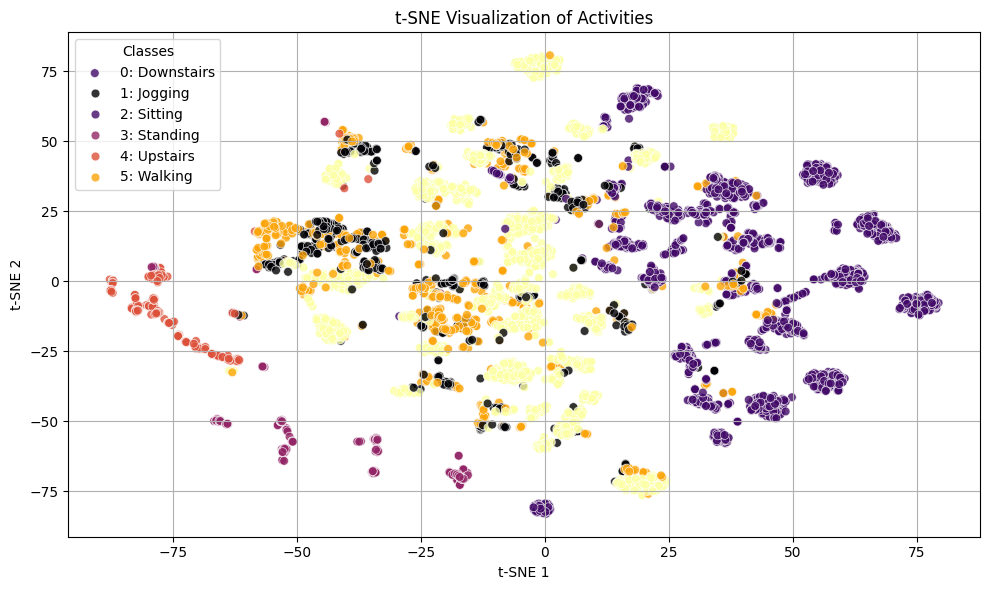
Exploratory data analysis

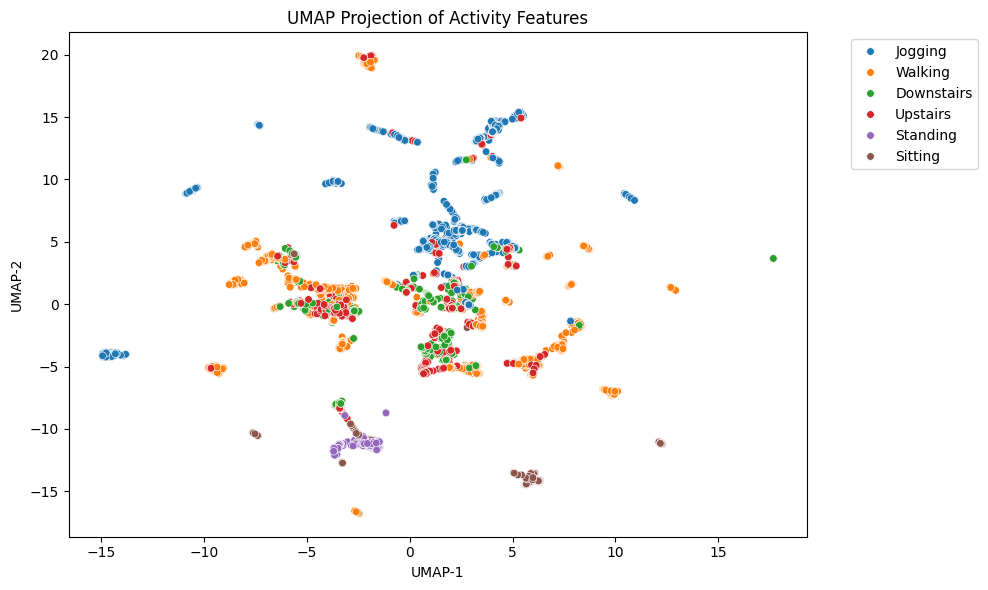
The types of features we extracted were geometric features amongst others from the signal data that we thought we could use to try to separate the classes more and give us a better chance of predicting the different classes better. These features included a median of each signal, interquartile ranges, skewness and kurtosis, range and the sum of the signal. We also extracted the euclidean norms of the means for each signal,the euclidean norm of the sum vectors and the axis ratios. This gave us a new dataset containing 69 columns. The list of the extracted features is as follows - "x-axis\_mean", "x-axis\_std", "x-axis\_max", "x-axis\_min", "x-axis\_median", "x-axis\_iqr", "x-axis\_kurtosis", "x-axis\_skew", "x-axis\_sum", "x-axis\_range", "y-axis\_mean", "y-axis\_std", "y-axis\_max", "y-axis\_min", "y-axis\_median", "y-axis\_iqr", "y-axis\_kurtosis", "y-axis\_skew", "y-axis\_sum", "y-axis\_range", "z-axis\_mean", "z-axis\_std", "z-axis\_max", "z-axis\_min", "z-axis\_median", "z-axis\_iqr", "z-axis\_kurtosis", "z-axis\_skew", "z-axis\_sum", "z-axis\_range", "timestamp\_count", "mag\_mean", "total\_energy", "x\_y\_ratio", "x\_z\_ratio", "y\_z\_ratio"

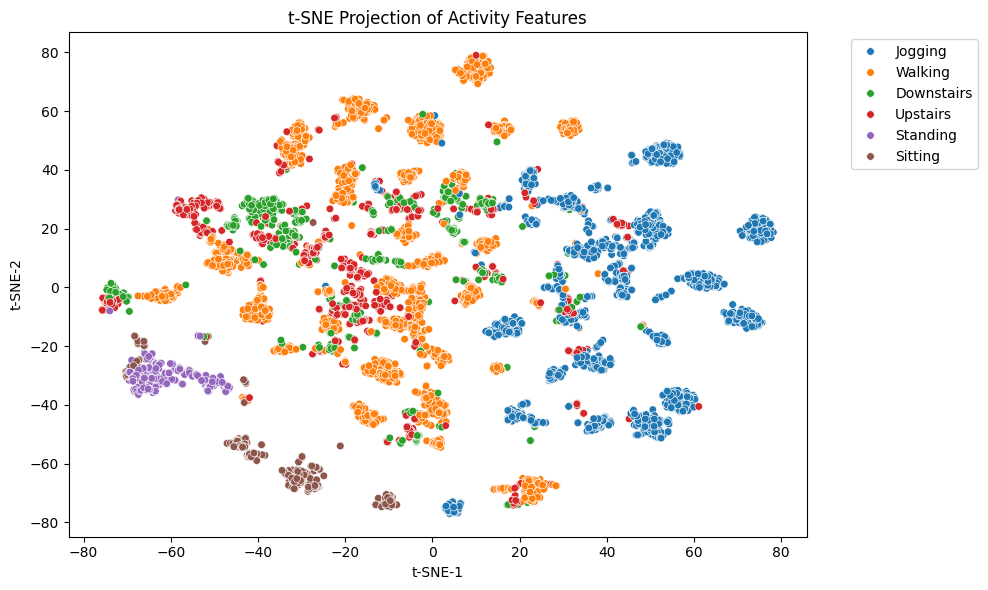
Before we extracted these features that we found worked best for our final model we created a different set of features, that were also geometric but did not include the range, sum of signals, euclidean norms, sum vectors or axis ratios. The full list of features of this dataset is as follows - ['user\_snippet', 'x-axis\_\_sum\_values', 'x-axis\_\_median', 'x-axis\_\_mean', 'x-axis\_\_length', 'x-axis\_\_standard\_deviation', 'x-axis\_\_variance', 'x-axis\_\_root\_mean\_square', 'x-axis\_\_maximum', 'x-axis\_\_absolute\_maximum', 'x-axis\_\_minimum', 'y-axis\_\_sum\_values', 'y-axis\_\_median', 'y-axis\_\_mean', 'y-axis\_\_length', 'y-axis\_\_standard\_deviation', 'y-axis\_\_variance', 'y-axis\_\_root\_mean\_square', 'y-axis\_\_maximum', 'y-axis\_\_absolute\_maximum', 'y-axis\_\_minimum', 'z-axis\_\_sum\_values', 'z-axis\_\_median', 'z-axis\_\_mean', 'z-axis\_\_length', 'z-axis\_\_standard\_deviation', 'z-axis\_\_variance', 'z-axis\_\_root\_mean\_square', 'z-axis\_\_maximum', 'z-axis\_\_absolute\_maximum', 'z-axis\_\_minimum', 'sma', 'resultant\_mean', 'corr\_xy', 'corr\_yz', 'corr\_zx', 'skew\_x', 'skew\_y', 'skew\_z', 'kurt\_x', 'kurt\_y', 'kurt\_z']. This feature extraction calculated a number of features that were not include in the original model such as; root mean square, absolute maximums, sma, resultant means and correlations.

For our exploratory data analysis, we began by using dimensionality reduction techniques such as PCA and t-SNE to visualise the data in a lower dimensional space for both the original data and the extracted features.

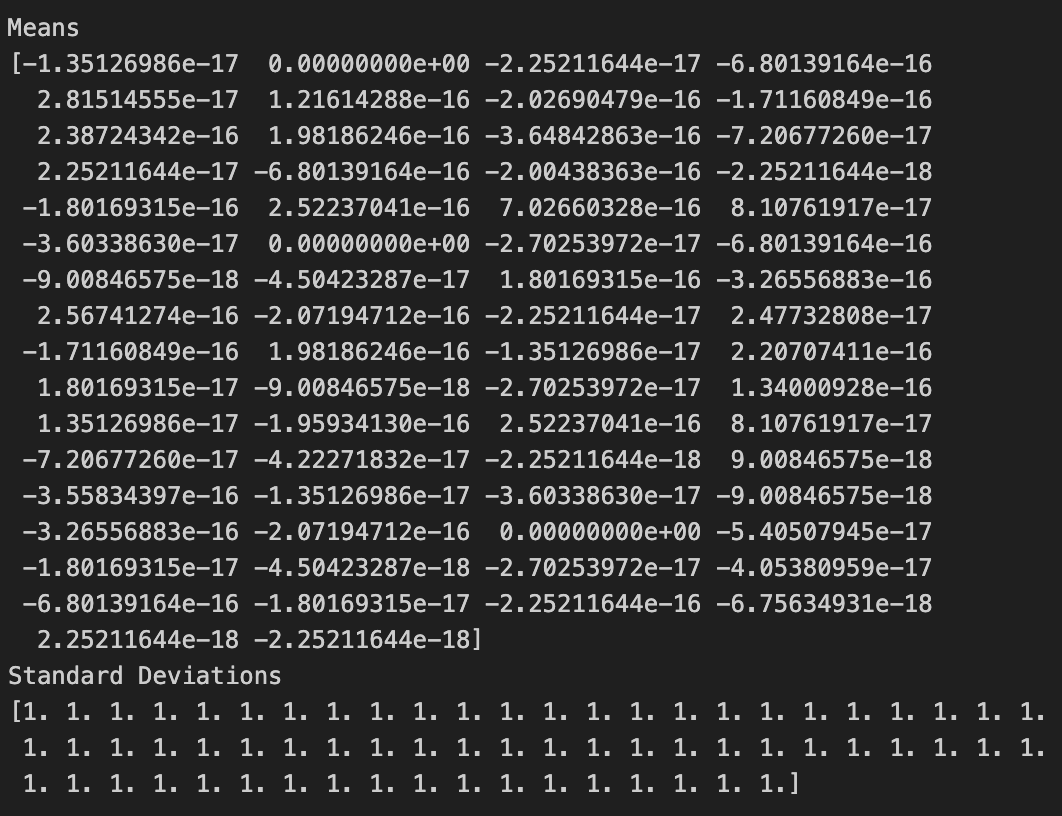








The data was normalised using z score normalisation to give the data a mean of 0 and standard deviation of 1. The normalised data was used in both our final model and non linear models that require normalisation such as svms and artificial neural networks. The normalisation of data is also necessary to fit data to one scale to avoid any biases that some models may have towards higher values.



We used a number of feature selection techniques based on the model that we were using. We used random forest feature selection by allowing it to pick which features it deemed most important for our final model based on the extracted features for our final model. However, we also used our early random forest models where we used just the original data with some of the extracted features, we got these results. We performed a number of searches dropping the value of the number of features to select in stages.

For the final model our 30 selected features were - ['y-axis\_\_standard\_deviation', 'y-axis\_\_variance', 'y-axis\_std', 'y-axis\_iqr', 'y-axis\_\_minimum', 'y-axis\_range', 'z-axis\_\_standard\_deviation', 'z-axis\_std', 'y-axis\_min', 'x-axis\_\_standard\_deviation', 'z-axis\_\_variance', 'x-axis\_iqr', 'z-axis\_skew', 'z-axis\_iqr', 'z-axis\_\_sum\_values', 'x-axis\_std', 'z-axis\_\_root\_mean\_square', 'mag\_mean', 'z-axis\_sum', 'z-axis\_\_median', 'z-axis\_\_mean', 'y-axis\_\_absolute\_maximum', 'total\_energy', 'z-axis\_median', 'x-axis\_\_variance', 'y-axis\_skew', 'z-axis\_mean', 'y-axis\_kurtosis', 'x\_y\_ratio', 'z-axis\_range']

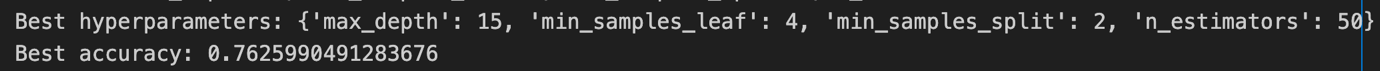
For our initial random forests our 15 features were - ['x-axis\_\_variance', 'y-axis\_\_standard\_deviation', 'y-axis\_\_variance', 'y-axis\_\_minimum', 'z-axis\_\_median', 'z-axis\_\_mean', 'z-axis\_\_standard\_deviation', 'z-axis\_\_variance', 'sma', 'resultant\_mean']

ML modelling

For our first attempt we tried a random forest with an exhaustive grid search classifier to identify the optimal hyperparameters. We did this for multiple iterations of random forest at first. We found that we were still getting a low accuracy score and a low AUC when plotting.

A graph showing the value of a training

AI-generated content may be incorrect.



We then decided to use a feature selection to extract the best features and filter out a lot of the noise that may not have been helping our model perform with the best accuracy. We used random forest The aim of this was to reduce the complexity of the model to improve the accuracy. By filtering out these features, our overall accuracy increased on some random forest iterations but decreased on others possibly due to overfitting with the hyperparameters being selected and the reduction of the number of parameters. We changed the number of features selected by trying various iterations and combinations and used a grid search after each one to find the optimal hyperparameters. We changed the hyperparameters of the random forest and the feature selection but found that our accuracy score was not improving.

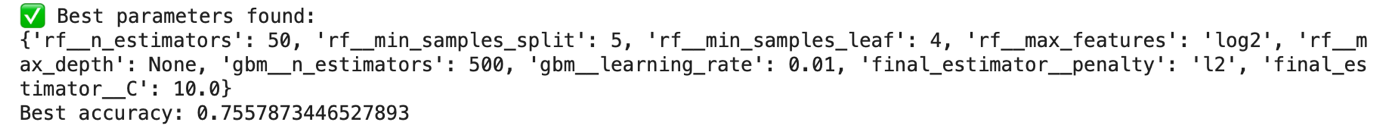
This is probably because of the data or the model itself overfitting, for this reason we continued to try other models and increasing their complexity.

After many attempts, we decided that GBM might improve our overall accuracy, even with the added risk of overfitting. We tried to integrate this with our best random forest and found that it did improve the model. For every iteration of the GBM we printed a confusion matrix to provide further insights.

A number and numbers on a white background

AI-generated content may be incorrect.

In an attempt to further optimise the score, we did a separate exhaustive grid search for the hyperparameters of the GBM. We found that it improved the calculated accuracy score but not the score on Kaggle. This led us to believe that overfitting was present in the model. We further did a grid search for the best hyperparameters with a GBM and a Rf and these were the results.



When we were satisfied that the GBM and Random Forest wouldn't improve our model anymore, we tried a different approach. Firstly, we standardised the features in order for it to be used in an SVM. Then once again performing a grid search to find the best hyperparameters for the SVM. We then ensured that the SVM uses the selected features for the prediction file. We ran this in the same code as our best performing random forest to compare the results. We produced a classification report for the random forest and SVM and presented them beside each other for comparison.

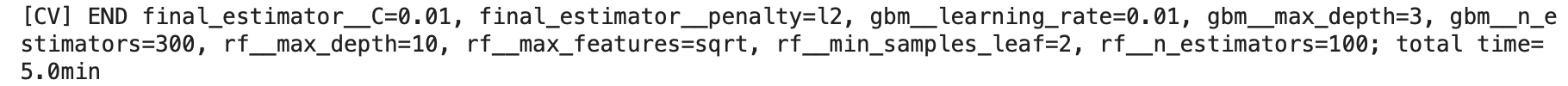
A screenshot of a computer screen

AI-generated content may be incorrect.

We found in every aspect the SVM was outperformed by the Random Forest including precision, recall and f-1 score. However, we concluded that this was due to overfitting and later discovered that the SVM gave us the best overall accuracy so far in Kaggle of 0.84.

From this we decided that we couldn't improve the model with the data we had provided. Therefore, we changed the way we imputed the data, now using the test data as validation data in training to aim to improve the accuracy. Using this in combination with our stacked model of RF and GBM, we found that this gave us the best score so far.

This was now our best and final model which gave us the highest accuracy. This was achieved by using a combination of, feature extraction and t-SNE for dimensionality reduction, a random search for both GBM and RF hyperparameters and finally evaluating the test data into validation during training. We are aware that a random search provides a wider range of hyperparameter searches at a faster speed as opposed to a grid search and here are the results of the random search.

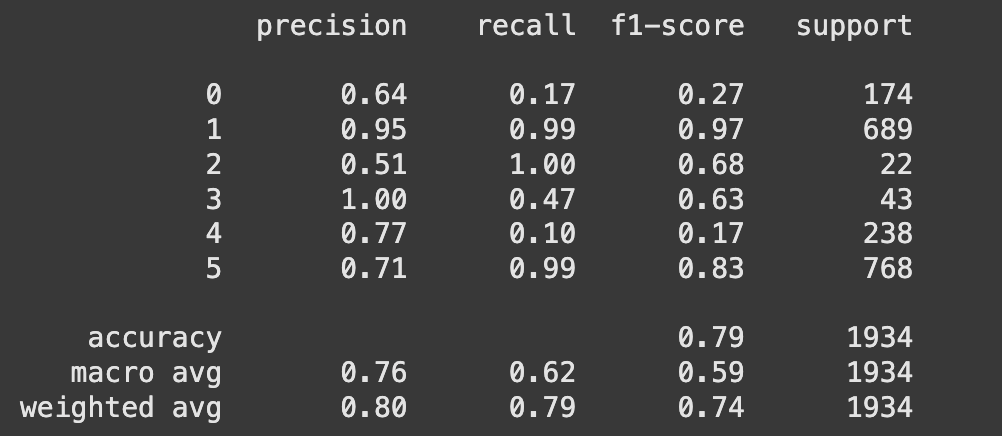


However, we understand that a random search isn't as effective as an exhaustive grid search that would find every combination. We used a random forest to select which features performed the best. Over many iterations this indexed each feature in order of importance and ran until it found the optimal number of features that needed to be used. Finally, we calculated a validation report for our final prediction. A screenshot of a computer screen

AI-generated content may be incorrect.

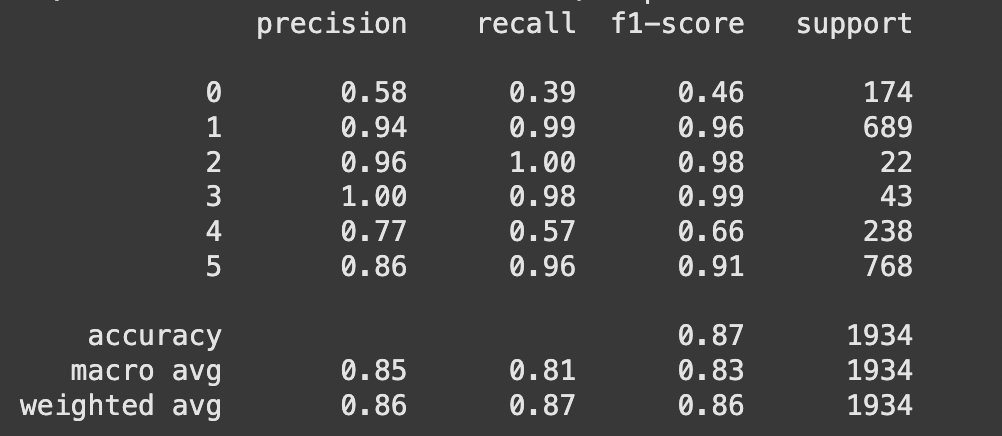
Xgboost

The results of the accuracy of the xgboost models were between 0.826 and 0.842. The highest score was achieved using the following hyperparameters from a random search of 30 iterations with a 3-fold cross validation. The model was run using the same extracted features that are used for the final model. The result of the best performing models hyperparameter search was - {'subsample': 1.0, 'reg\_lambda': 2, 'reg\_alpha': 0.1, 'n\_estimators': 100, 'max\_depth': 3, 'learning\_rate': 0.01, 'gamma': 0.1, 'colsample\_bytree': 0.8}. We also produced a classification report below showing the overall accuracy of the model as well as metrics of how the model performed in predicting each class individually.



Artificial neural network

The results of our artificial networks resulted in Kaggle scores that ranged between 0.81 and 0.825 on the public leaderboard using the same features that are used in the final model. We used a keras tuner random search for hyperparameter tuning of 20 trials using 30 epochs per trial. The resulting hyperparameters chosen by the random search were; {'num\_layers': 3, 'units\_0': 192, 'activation': 'relu', 'dropout\_0': 0.4, 'learning\_rate': 0.008223713071187294, 'units\_1': 128, 'dropout\_1': 0.4, 'units\_2': 64, 'dropout\_2': 0.30000000000000004}. The resulting hyperparameters were used to run the model that produced the comparison matrix below. This particular model received a Kaggle score of 0.825.



Discussion

The results of our exploratory analysis show the non-linearity of our data, and this can be seen in the PCA and t-SNE plots of the original data. A number of the classes involving movement like walking and jogging are clumped together in the PCA plot showing no separation whereas there is more separation between classes like sitting and standing. Overall, it is difficult to distinguish between the classes in the PCA plot. The t-SNE plot for the original data on the other hand is more informative and shows more separated classes most likely due to t-SNE preserving the local structure of the data and PCA’s linear nature in dimensionality reduction. The t-SNE plot still shows little separation between walking and jogging which could be down to the similarity between the two in real life. Sitting and standing are separated clearly to the left-hand side of the plot and tightly grouped. Downstairs is an activity that is somewhat separated to the right-hand side of the plot, but it does overlap slightly with walking and jogging. The result of this analysis was what we based our model selection on. We decided to choose models that could handle non-linear data some being ensemble methods like random forest, GBM, SVM, xgboost and we also thought that an artificial neural network may be able to make good connections in the relationships in the non-linear data.

Since we found that our data is non-linear, we decided to visualise our extracted features in lower dimensions to see if the extracted features would help improve the performance of our models. The results of the umap show similar results in that sitting and standing are separated well in tight clusters, jogging is also separated clearly into a number of tight small clusters. Walking, standing and upstairs are separated throughout the centre of the plot and overlap slightly in places. The results of t-SNE are similar in that jogging was separated into a number of small tight clusters with some slight overlap between some of the classes that are spread across the centre like walking, downstairs and upstairs. Again, sitting and standing were separated well into small clusters at the bottom of the plot. These plots suggest that the model may struggle in classifying some of classes especially upstairs and downstairs but will most likely fair better with classifying the classes that showed clearer clusters and separation.

The extracted features that we extracted for use in the final model was done as a result of our inability to improve model performance in our early models, this was probably due to them not necessarily adding much to the model and in the results of our feature selection using random forests we found that a majority of the features were chosen but ultimately we had to extract more to try and improve our results. The additional features that we extracted second time around helped to boost our model performance and we also found that a number of the extracted features were selected when using random forest feature importance. The reason for using feature selection was an attempt to remove any features that may not contribute to the performance of the model, this is due to the fact that feature selection reduces model complexity.

The normalisation of our data was necessary as it is not needed for some models such as random forest and GBM. However, normalisation is necessary for other models such as artificial neural networks, xgboost and SVM. When running our models using normalised data, we found that the model performance improved, on Kaggle in particular, which could be due to the removal of potential bias where larger values may be treated differently to smaller values in some models.

In our earlier Random Forest and GBM models we experienced overfitting due to the limited features we provided in the data. This can be seen in the first training curve provided in the ML modelling section. The training data accuracy is stable around 1 throughout, whereas the validation set increases up until around 0.75 where it plateaus. This is clear evidence of overfitting. We tried multiple iterations of each before using a hyperparameter search, although this did not increase our accuracy due to the lack of data. The initial GBM boosted our accuracy from 0.76 in RF to 0.93, but this wasn’t shown in Kaggle as our highest accuracy was around 0.84. This is further evidence that the models fit on the restricted data were overfitting. This was the same for the SVM as the data we used was limited.

After finding that our models were not improving, we went back to perform more feature extraction to try and get more data for the model to train off of and also see if we could extract more meaningful features that would improve performance. The use of these features when implemented into our xgboost, neural network and stacked model helped to improve our Kaggle score. We also made the realisation that giving the model more data would help us improve performance by utilising the test set for use during validating the model before making final predictions. This is due to the fact that giving the model more data will usually help to improve performance and can reduce the risk of overfitting, it also allows us to compare results better by having validation accuracy as a metric that we can look to improve.

We also made use of artificial neural networks and the results of running these models was similar to all of our other ensemble models. This model like all of our others struggled to predict the upstairs and downstairs class. This suggests that there may be an issue with our data that we need to address if we were to make improvements to this assignment in the future. However, we believe that with the results that we found from our approach in our final model we believe that the way forward is by using stacked ensemble models. An improvement that we could make to our neural network would be a grid search to find the optimal hyperparameters of our neural network or potentially look at other neural network architectures.

Our final model used this approach in combination with stacking a random forest and a GBM. We did use a random search for the hyperparameters of these as a grid search will have taken too long. If given more time a grid search would help find the exact hyperparameters to optimise the model. By running more random searches, this will be more rigorous than just a single random search and will require less computational time than a grid search.

Initially we focused too much time and attention to our earlier models under the illusion that our parameters weren’t optimal, although the data we were using for the models was incorrect. When we began to stack the algorithms, we noticed a significant increase in our Kaggle score of around 0.03. This leads us to believe that by stacking different models will be the best approach in future data analysis. For example, our base xgboost model outperformed the others on Kaggle. Unfortunately, we only realised this when we were pressed for time. If we could later stack our xgboost with our random forest, we believe this would give us an improved score.

Most of our models use logistic regression as a final layer in our stacked model. Even though logistic regression is an algorithm that focuses on linear data we still found it to be an effective final estimator due to the increase in our model performance. However, by using a more suitable model that accommodates non-linear data, this would open a new avenue to potentially improve the model further.

Reviewing our validation accuracy scores we can see that the model was poor in identifying upstairs and downstairs but performed well when classifying standing, sitting and jogging. Upstairs did fair well with impressive scores for precision but had a low recall and f-1 score. The precision, recall and f-1 score was significantly lower for downstairs and upstairs compared to the others. For example, the precision of downstairs was 0.4 compared to 1.0 in the standing class. This is concurrent throughout the validation scoring, where standing performed the best. This could be a result of overfitting and should be improved by possibly selecting different features. This can be linked to the exploratory analysis of the t-SNE and umap, where we thought we may struggle with prediction of upstairs and downstairs due to the lack of separation in classes and no real clusters. Sitting can be seen as a clear cluster compared to downstairs and upstairs where a lot of overlapping was present.