Predicting Activity
Based on Sensor Data

Vira Oetterli,Tillman Jex Schäuble, Davide Attebrant Sbrzesny

DM1590 Machine Learning for Media Technology
2023, KTH, Stockholm

Table of Contents

Abstract

This project explores using machine learning (ML) models to predict the movement activities of humans. The data was obtained from the UCI database and consists of motion capture data acquired from sensors worn by human participants.
Multiple supervised and unsupervised ML models were used and the data was further augmented with feature engineering. Through our work we critically analyse the usefulness of our approaches, the quality of the data and present cases for future work that will interest others looking to implement ML models for movement prediction applications.

Background

We used the Localization Data for Person Activity (UCI 2010)dataset. It was created for use in the research paper *An Agent Based Approach to Care in Independent Living* (Kaluža et al. 2010) to investigate the creation of a multi-agent system for the care of elderly people living at home on their own, with the aim to prolong their independence. With this system, software could be developed to act as an alarm if an elderly person has fallen and cannot get up for example.

The dataset consisted of x, y, z coordinates of 4 sensors worn by participants as they moved around a 25 sqm room. The sensor data was captured as a stream of data (i.e. sequential

time series) and labelled according to what the participant was doing at the time (walking, sitting, lying down, etc).
There were 5 participants in total, and each participant was tracked 5 times, leading to 25 sets of movement data containing various movement activities.

	sequence_name	tag_identificator	timestamp	date	x-coordinate	y-coordinate	z-coordinate	activity
0	A01	ankle_left	633790226051280329	2009-05-27 14:03:25.127	4.062931	1.892434	0.507425	walking
1	A01	chest	633790226051820913	2009-05-27 14:03:25.183	4.291954	1.781140	1.344495	walking
2	A01	belt	633790226052091205	2009-05-27 14:03:25.210	4.359101	1.826456	0.968821	walking
3	A01	ankle_left	633790226052361498	2009-05-27 14:03:25.237	4.087835	1.879999	0.466983	walking
4	A01	ankle_right	633790226052631792	2009-05-27 14:03:25.263	4.324462	2.072460	0.488065	walking
...	...	...	...	...	...	...	...	...
164855	E05	ankle_right	633790146419554374	2009-05-27 11:50:41.957	3.209474	2.044571	0.062902	walking
164856	E05	ankle_left	633790146419824669	2009-05-27 11:50:41.983	3.386878	2.004729	0.395161	walking
164857	E05	chest	633790146420094965	2009-05-27 11:50:42.010	3.188895	1.915717	1.353087	walking
164858	E05	ankle_right	633790146420635550	2009-05-27 11:50:42.063	3.150169	1.931164	0.055037	walking
164859	E05	ankle_left	633790146420905847	2009-05-27 11:50:42.090	3.209994	1.939577	0.364777	walking

original dataset

Initially we thought that each datapoint was comprised of each sensor's {x, y, z} coordinate as well as a timestamp. With the data structured in such a way, we had intended to create interesting features such as distance in z-coordinate between belt and chest, to infer what position someone was in. For example, if the person was standing up, the two sensors would be aligned on the z-axis, while if they were lying down, they would be aligned on the x-axis or y-axis (depending on their orientation).

However as we looked closer at the time stamps, we realised that each sensor was in fact being logged sequentially in time, in increments of ~100ms. The dataset then called for time series analysis, the difficulty of which we were warned about. After some preliminary research into the subject, we conceded that it was too much of an undertaking for this project and if we went about trying to achieve it, other aspects of the project would lose attention and suffer in quality.

We returned then to the original question; What can we do with the dataset, considering tackling time series analysis is not an option?
We decided to stick to the original plan of creating an activity classifier, and to do this by engineering new features from the dataset that could present useful relationships with which we could work (see the 'Feature Engineering' section).

Methods

Data Splitting

As mentioned, our dataset contained the movements of 5 individuals (A, B, C, D & E), each undergoing 5 tracking sessions. This provided us the opportunity to perform a manual split of our dataset (as opposed to a built-in `train_test_split` method).

We did this by taking all of participant A's sequences, A01-A05, as the test data, and the remaining sequences as the training data, resulting in an 80/20 split.
There are bound to be several, nearly identical datapoints in each sequence since the sampling frequency is so high. Including what are essentially almost duplicates in the test and training set would likely give cause to a false high accuracy.

```
# All A01-A05 sequences are selected (encoded 0-5). Roughly 20% of our data.
train_data = df[df['sequence_name'].isin(['A01', 'A02', 'A03', 'A04', 'A05'])]
test_data = df[df['sequence_name'].isin(['B01', 'B02', 'B03', 'B04', 'B05', 'C01',
                                         'C02', 'C03', 'C04', 'C05', 'D01', 'D02',
                                         'D03', 'D04', 'D05', 'E01', 'E02', 'E03',
                                         'E04', 'E05'])]
```

Feature engineering

The opportunities for feature engineering came from navigating around the time series aspect of the dataset.
The time stamps followed the format of HH:MM:SS:Sss. We explored the idea of removing the subsecond range from the entire dataset, which would allow us to group coordinate measurements in second intervals. We could then engineer a new feature based on, for example, the mean {x, y, z} coordinate of all sensors within that one second time frame. However Bob suggested a better direction, which was to look at interpolating the {x, y, z} coordinates to generate new features.

We started by creating columns for each coordinate and tag, a total of 12 new columns. We filled in the blank spaces between values using the built in method for data frames. We took advantage of the fact that we had specific times for the datapoints by setting the method to "time" which takes time between points into consideration when interpolating new data values. We used back fill and forward fill to estimate the first and last values for each tag.

By creating new features with interpolated values we are using all the information we have to make predictions at a given time. There are risks associated with doing this. The method is sensitive to prolonged pauses in one sensor and will create values that are far from their true locations, which might negatively impact the ability to accurately train and predict on the data. However, the sensors all sample at a very high rate which reduces the risk of this happening.

	sequence_name	timestamp	date	activity	chest_x-coordinate	chest_y-coordinate	chest_z-coordinate	belt_x-coordinate	belt_y-coordinate	belt_z-coordinate	ar_x-coordinate	ar_y-coordinate
0	A01	633790226051280329	27.05.2009 14:03:25:127	walking	4.291954	1.781140	1.344495	4.359101	1.826456	0.968821	4.324462	2.07246(
1	A01	633790226051820913	27.05.2009 14:03:25:183	walking	4.291954	1.781140	1.344495	4.359101	1.826456	0.968821	4.324462	2.07246(
2	A01	633790226052091205	27.05.2009 14:03:25:210	walking	4.309984	1.777778	1.333371	4.359101	1.826456	0.968821	4.324462	2.07246(
3	A01	633790226052361498	27.05.2009 14:03:25:237	walking	4.328015	1.774416	1.322246	4.362803	1.841832	0.918870	4.324462	2.07246(
4	A01	633790226052631792	27.05.2009 14:03:25:263	walking	4.346046	1.771054	1.311121	4.366505	1.857208	0.868920	4.324462	2.07246(
...	...	...	...	...	...	...	...	...	...	...	...	...
8888	E05	633790146419554374	27.05.2009 11:50:41:957	walking	3.140489	1.936017	1.423725	3.359577	2.027247	1.070798	3.209474	2.044571
8889	E05	633790146419824669	27.05.2009 11:50:41:983	walking	3.164692	1.925867	1.388406	3.359577	2.027247	1.070798	3.189705	2.006769
8890	E05	633790146420094965	27.05.2009 11:50:42:010	walking	3.188895	1.915717	1.353087	3.359577	2.027247	1.070798	3.169937	1.968966
8891	E05	633790146420635550	27.05.2009 11:50:42:063	walking	3.188895	1.915717	1.353087	3.359577	2.027247	1.070798	3.150169	1.931164
8892	E05	633790146420905847	27.05.2009 11:50:42:090	walking	3.188895	1.915717	1.353087	3.359577	2.027247	1.070798	3.150169	1.931164

Interpolated dataset

Outlier Detection

By plotting the sensor coordinates in 3D space, assumptions could be made about the room. The mean of the z-coordinate, in the activity walking, was 1.27 for the chest sensor and 0.15 for the left ankle sensor. This implies it would not be unrealistic to assume the z-coordinates were measured in meters with the floor as the zero-point. However, if that was true, we had some points deviating with unrealistic measurements. E.g. There were points recorded at 2 meters below the floor while lying and points recorded at 2 meters above the floor while walking from the ankle sensors. Therefore it was evident to assume these points were inaccuracies in the sensor recordings.

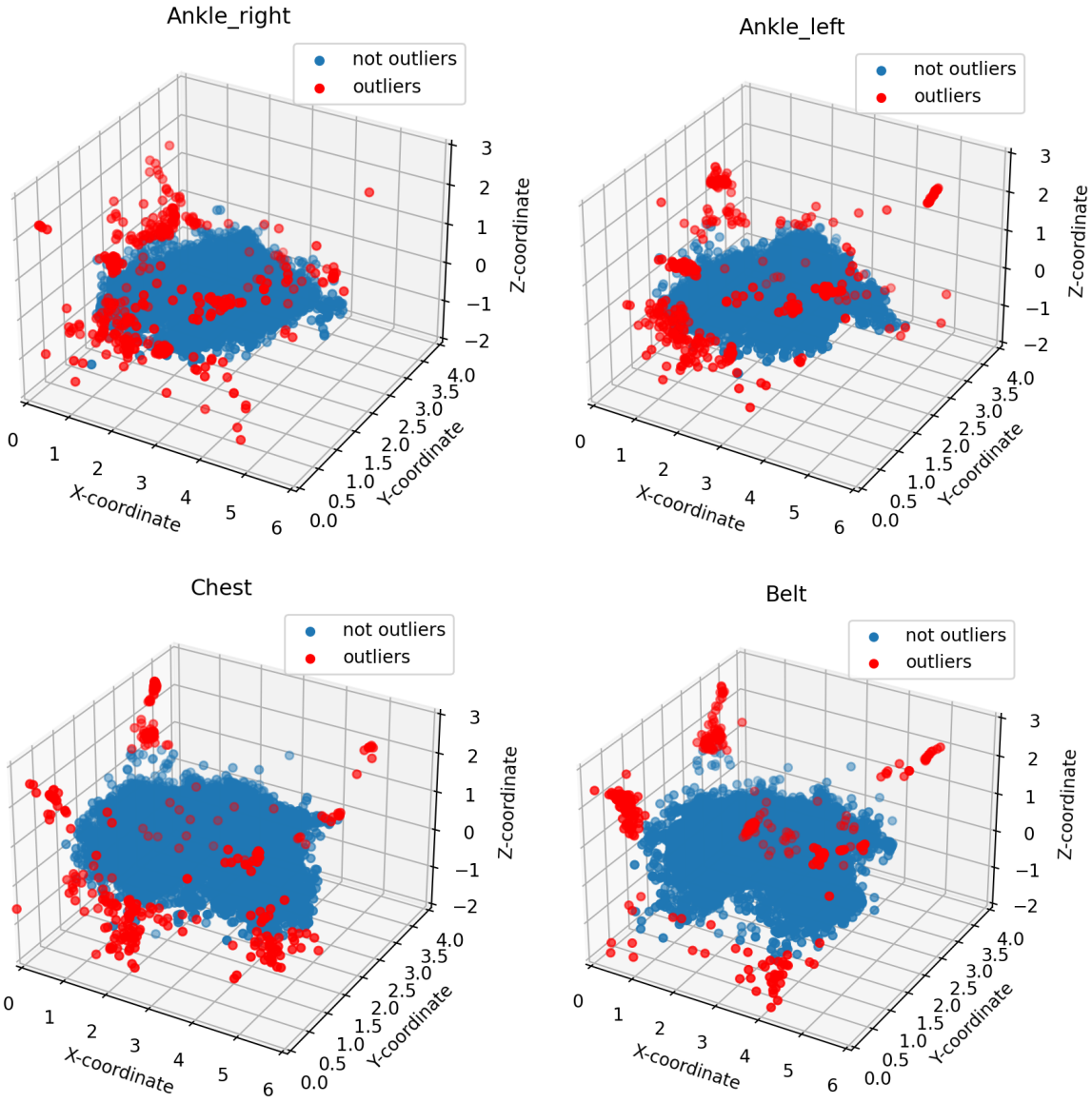
Outlier detection was used to filter out these points. This unsupervised method aims to identify any samples that significantly deviates from the rest, classifying them as outliers. The model we used is called Isolation Forest, which is built up of multiple decision trees. Each tree works by randomly selecting a feature and a split value for that feature, and then recursively partitioning the dataset into smaller subsets using this split. Outliers are identified as data points that require fewer partitions to be isolated from the rest of the data, as they have unusual values for one or more features. (Liu, Ting and Zhou, 2008)

One of the advantages of Isolation Forest compared to other outlier detection models is that it takes multiple features into consideration, which suits our 3 dimensional dataset. Therefore the parameter for the maximum amount of features when training the model was set to 3. Another advantage is that it detects outliers specifically, as opposed to detecting which points are normal and then stating that the rest are outliers. This requires less computational power since the normal points are detected at a greater depth than the outliers in the trees.

Each point receives an anomaly score between 0 and 1, reflecting the degree of its abnormality. (Liu, Ting and Zhou, 2008) The parameter contamination is the threshold for

how many percent of the data should be classified as outliers. Not wanting to lose a big chunk of the data, a threshold of 1% was selected. The other parameters were set to their default values.

Each sensor was segregated when performing the outlier detection so that they would not detect points from other sensor coordinates as outliers. After the model had classified the outliers, a new dataframe containing only outliers was made as well as a data frame containing only normal points. The dataframes without outliers would then be used to perform supervised learning. A scatterplot for each sensor was plotted with both the outliers and normal points to get a visual understanding of the outliers.



Scatterplots of the Outlier Detection results

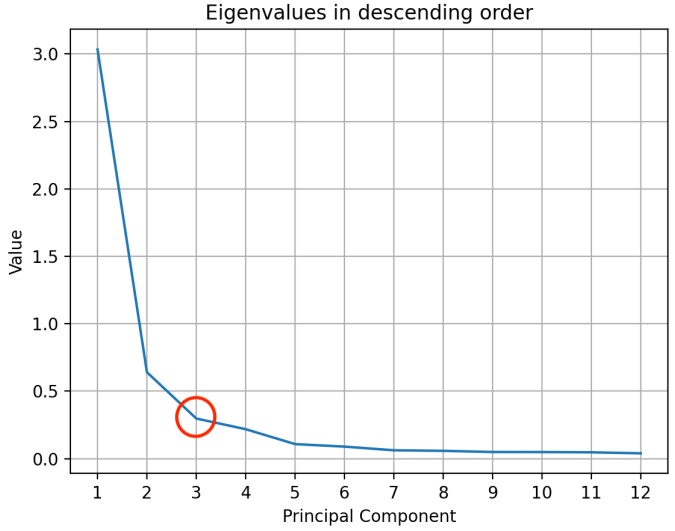
The resulting scatterplots from the outlier detection, clearly indicate that the most extreme outliers were detected and could be removed from the data with success.

PCA

Interpolating the sensor coordinates increased the dimensionality by a factor of 9 (3 coordinates x 3 sensors), resulting in heavier demands on the computer. Therefore, PCA was conducted to reduce the number of dimensions but without losing too much information. PCA is a dimensionality reduction method which projects the data onto each of its eigenvectors. The new dimensions are called Principal Components (PCs), and the number of PCs are chosen to maximize the variance of the data.

To perform PCA on the grouped sensor coordinates, the data was centered around the origin, and the covariance matrix of the centered data was eigen decomposed. The resulting eigenvalues represent the variance explained by each PC, with higher eigenvalues indicating greater variance. Eigenvectors indicate the direction of maximum variance. To ensure the maximum amount of variance is obtained, the eigenvalues with its corresponding eigenvector were sorted in descending order.

The PCs were obtained by projecting the centered data onto the eigenvectors. The PCs are a linear combination of the original features that capture the maximum amount of variation in the data. To decide how many Principal Components to keep, a plot of the eigenvalues and corresponding component was made, to see if there was a distinct elbow point where most of the variance was retained. The cumulative variance was also calculated to get a more precise value of how much variance the PCs contained.



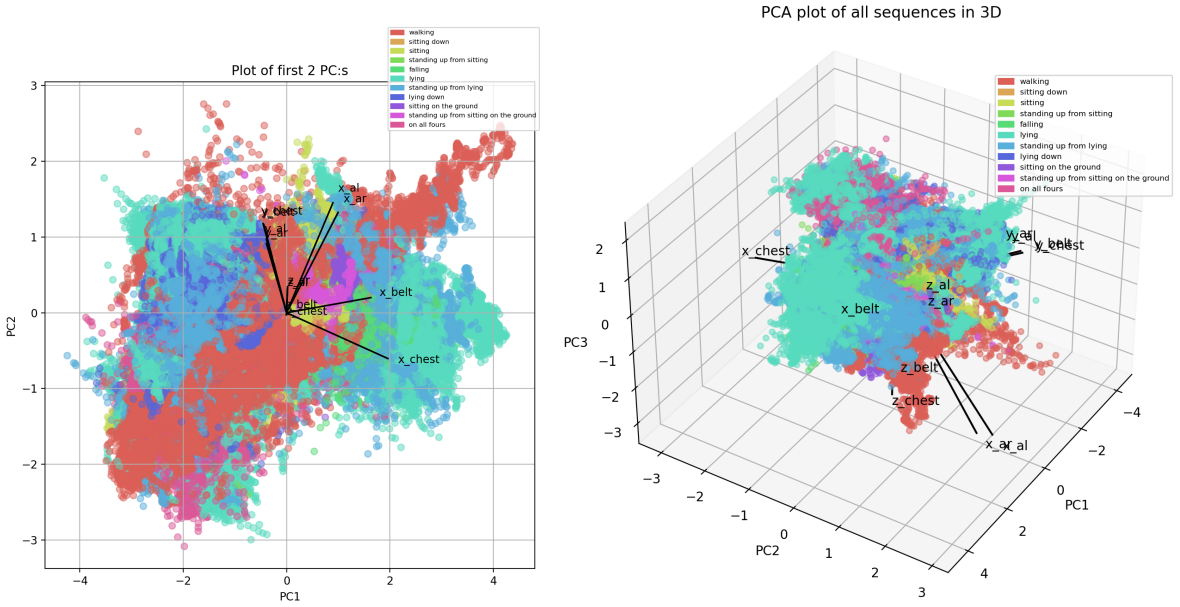

```
Cumulative variance in the first n eigenvectors:
n = 1 0.647
n = 2 0.7836
n = 3 0.847
n = 4 0.8935
n = 5 0.9165
n = 6 0.9354
n = 7 0.9486
n = 8 0.9609
n = 9 0.9713
n = 10 0.9817
n = 11 0.9917
n = 12 1.0
```

Eigenvalues and cumulative variance

There was an elbow point in the plot at 3 components, which contained 84.7% cumulative variance. Although 84.7% was a high amount of variance retained in only 3 components, the goal was to keep as much of the information as possible while still reducing dimensions. Therefore 9 components were also selected, with a cumulative variance of 97.1%. The high amount of variance retained shows that the dimensionality reduction was successful.

To get a better visual understanding of the PCA, the first 2 PCs and the first 3 PCs were plotted, with vectors from the origin to the points corresponding to each coordinate. Even

though it is not unsupervised learning, the classes were added as different colors to the plot, to determine if the classes could be distinguished.



Scatterplots of two and three PCs

By analyzing the principal component plots, it is evident that the x-coordinates have the highest variance. This is most likely due to the data not being scaled. The x-axis has the largest range of values, suggesting it covers the longest wall in the room. In contrast, the z-coordinate exhibits minimal variance in the first two PCs but shows a stronger impact on the third. This could be improved by scaling the x, y and z-coordinates so that their samples are in the same range.

However, despite the successful dimensionality reduction, the PCA plot did not show clear separations between the classes in either 2 or 3 PCs, indicating that PCA might not be the best choice for this particular dataset.

Lastly the PCs were made into a new dataframe, which was then used as features in the supervised learning models.

KNN (K Nearest Neighbour)

KNN is a supervised machine learning model that classifies data by comparing the distance between the feature values in a datapoint with those in the training set. It finds the closest number of neighbours (‘K’) and looks at their labels. It classifies as the most common label seen. K is a parameter that can be set by the user (Guido and Müller 2016).

We chose to use KNN as it is considered a good starting point for supervised machine learning. It is an intuitive method to understand, especially when using only one sensor with three coordinates as it then, quite literally, finds the nearest points in space. Furthermore, we

had rather few features and we did not have a sparse dataset, which would have been poor for the performance of KNN (Guido and Müller 2016).

We started by splitting our data into a test and a training set. Since each of the 25 sequences (A01 to E05) was sampled many times each second, there were bound to be many datapoints which were nearly identical. Including datapoints from one sequence in both test and training set would have likely given a misleadingly high accuracy. Furthermore, all sequences performed by the same person started with the same letter. We wanted to test how well our model generalises when predicting activities by a new person. We therefore decided to use sequences from one person as test data and the other four as training data. The sequence was selected at random.

When testing for the best parameter value for K, a similar approach was used. Each of the four groups of sequences (B01-05, C01-05, D01-05, E01-05) was used once as a test and three times as training data when searching for the best parameter value. Initially, every tenth k-value was tested to find the rough area of the best generalisation. When that area was identified, every k-value in the area was tested. This was done to save time. The distance parameter was selected as ‘euclidean’ as it was most intuitive when comparing physical distances for sensors.

The next step was to train a model on the entire training dataset and then test it on the test set. Evaluation was performed using an accuracy report and confusion matrix. The purpose was to see how well it predicted different activities as well as understand what errors it was prone to make.

These steps were performed on several different datasets. The first dataset, with chest sensor information only, was used as a baseline. The second set used the twelve dimensional dataset created through interpolation of new datapoints and the third was a dataset with the same twelve features, but without outliers. This was used to see whether the removal of erroneous datapoints might improve the model. A fourth dataset that used the output from the PCA model, the three principal components that contained the most variance were used.

However, when analysing our dataset and the nature of its creation we soon realised that our methods did not quite align with our purpose. Our aim was to predict a person's position based on one datapoint. However, how is a single datapoint supposed to know whether someone is rising from sitting, or sitting down from standing? This requires more information. We therefore tried to remove all activities that required such information. Activities such as “walking” and “lying” were kept. Furthermore, some activities were very rare and were also removed. One such activity was “standing on all fours”, which rarely occurred.

Upon further consideration, we realised that our models, especially when only using data from one sensor, were looking at the absolute location of a person and finding the nearest spot in the training set. This means that it looks at a persons location in the room and, based on that, tries to predict the persons activity. This would make sense if we wanted to analyse a room and draw conclusions such as ‘*there is probably a bed in that corner, as most datapoints collected there are labelled as lying’*. But we wanted to predict activity based on how the sensors related to each other. We tried to create a new dataset that contained new

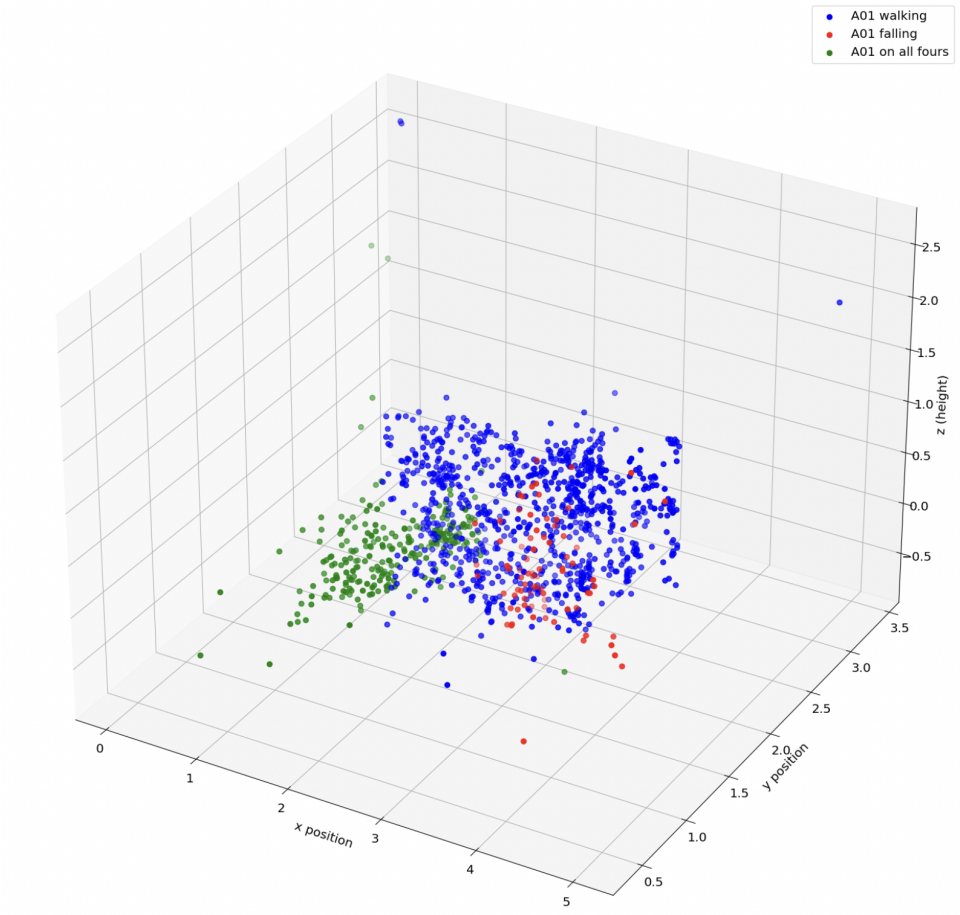
features. Euclidean distance between sensors as well as difference in z-coordinate between sensors. These features were created based on ‘expert knowledge’ from us. This was, however, done very late in the project and its purpose was merely to find areas of improvement for future studies. As such, most of the report does not deal with this dataset.

SVC (Support Vector Classifier)

SVC is a supervised machine learning model that aims to achieve classification by finding the clearest divisions between collections of data and then maximising the division. It has four different strategies to divide the classified data, linear, rbf, polynomial and sigmoid.

Each of these strategies, called *Kernels*, operate on the principle of translating the dimensionality of the input dataset into a higher order and fitting a decision boundary between the data. Each Kernel has a different shape. Achieving a higher accuracy is highly dependent on choosing the best Kernel shape for the dataset as well as other variables known as *Hyperparameters*.

We identified SVC as being a good choice of model as it can be tuned to handle outliers (by varying the ‘*C’* value), can handle time series data and operates like a weighted nearest neighbour model. Upon initial observation of our data, all of these aspects suited our dataset well as our data had quite a few outliers, was time series based and showed fairly strong inherent grouping in the raw data, as seen below.



Scatter plot showing three activities from the A01 sequence

The disadvantages of SVC are however that it is very processor intensive and is difficult to understand or explain *why* a particular kernel and its hyperparameters are the best fit. This is due to the confounding nature of fitting a 'shape' in multidimensional space, which naturally has no practical way to be shown visually.

We performed the following sequence to train and test the SVC:

1. Create the manual test / train split (80 / 20) outlined previously
2. Feature scaling*
3. Systematically testing different combinations of kernels and parameter values using four different splits of the training data for each model to train and test it on. Accuracy score was used as a measurement of the different models' performances.**
4. Model training based on cross validation results
5. Check for overfitting***
6. Check null accuracy score
7. Generate confusion matrix (see *Results* section)
8. Generate classification report (see *Results* section)

* It is recommended for SVC models as if it isn't, the features with the largest range will completely dominate in the computation of the kernel matrix (Tokuç 2022), which is the underlying algorithm used to calculate the classifiers.

** To obtain the most appropriate SVC model and its hyperparameters we used cross validation, specifically k-folds and grid search. This allowed us to programmatically uncover the best model and hyperparameters combination.
Cross validation results: {kernel = rbf, c = 1.0, gamma = auto}

```
# Set up a range of hyperparameters to test.
# these are the parameter combinations that we want to test for the svc
param_grid = {'C': [1.0, 10.0, 100.0],
              'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
              'gamma': ['auto', 'scale']}

# input all established arguments along with the model we want to test them with
# n_jobs=-1 requests to use all available cpu cores for the process
# uses k-fold by default
grid_search = GridSearchCV(SVC(), param_grid, n_jobs=-1)

# Fit the grid search object to the training data
# run the cross validation against the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters and corresponding score
print("Best hyperparameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

code for cross validation

*** We did observe some overfitting as shown below and found that a 70/30 fit achieved 4% higher mean accuracy than the 80/20 split, while also minimising overfitting. We also tried a 60/40 split, but this did not yield much change in accuracy results or other metrics, and therefore led us to stay with a 70/30 split. The results in the following sections for SVC are therefore based on a 70/30 split, while the other models use the original 80/20 split.

```
# check data for overfitting
print('Training set score: {:.4f}'.format(svc.score(X_train, y_train))),
print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

```
Training set score: 0.9292
Test set score: 0.6877
```

overfitting being observed with 80/20 split

```
# check data for overfitting (70 / 30 split)
print('Training set score: {:.4f}'.format(svc.score(X_train, y_train))),
print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

```
Training set score: 0.9159
Test set score: 0.7303
```

overfitting minimised with 70/30 split

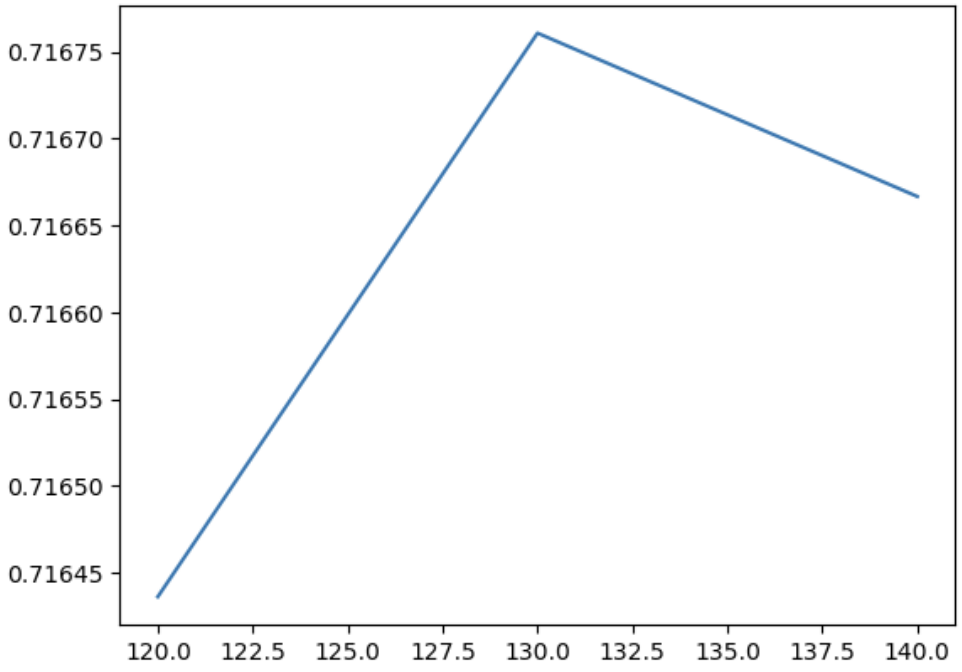
```
# check data for overfitting (60 / 40 split)
print('Training set score: {:.4f}'.format(svc.score(X_train, y_train))),
print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

```
Training set score: 0.9126
Test set score: 0.7432
```

overfitting minimised further with 60/40 split (but no marked change in performance results)

Results

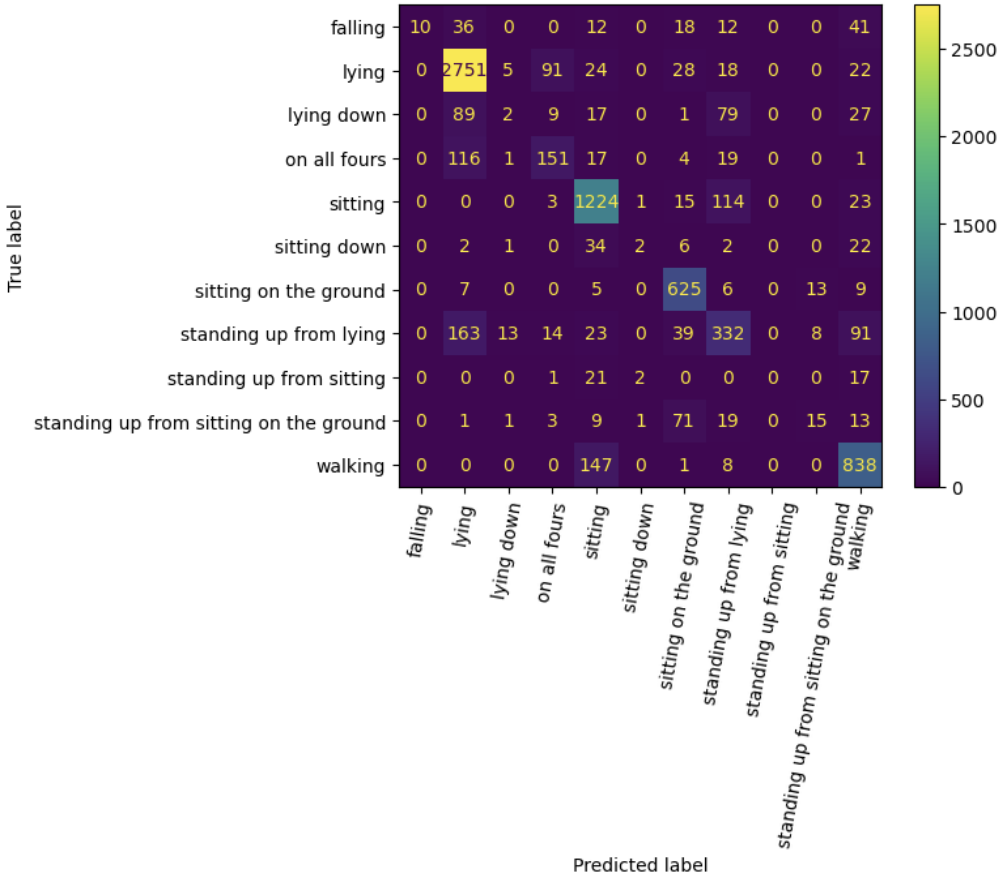
KNN




```
Best K is:  130  has value:  0.7167606926719783
```

K-value testing only chest sensor.

	precision	recall	f1-score	support
falling	1.00	0.08	0.14	129
lying	0.87	0.94	0.90	2939
lying down	0.09	0.01	0.02	224
on all fours	0.56	0.49	0.52	309
sitting	0.80	0.89	0.84	1380
sitting down	0.33	0.03	0.05	69
sitting on the ground	0.77	0.94	0.85	665
standing up from lying	0.55	0.49	0.51	683
standing up from sitting	0.00	0.00	0.00	41
standing up from sitting on the ground	0.42	0.11	0.18	133
walking	0.76	0.84	0.80	994
accuracy			0.79	7566
macro avg	0.56	0.44	0.44	7566
weighted avg	0.75	0.79	0.76	7566



Chest-sensor only. K-value = 130

For further values, see appendix.

SVC

	precision	recall	f1-score	support
falling	0.24	0.15	0.19	2204
lying	0.86	0.85	0.85	37780
lying down	0.25	0.20	0.23	4605
on all fours	0.38	0.40	0.39	3374
sitting	0.88	0.86	0.87	19316
sitting down	0.14	0.08	0.10	1242
sitting on the ground	0.93	0.41	0.57	7849
standing up from lying	0.43	0.46	0.44	13934
standing up from sitting	0.13	0.02	0.03	1049
standing up from sitting on the ground	0.20	0.53	0.29	2038
walking	0.83	0.95	0.88	25588
accuracy			0.73	118979
macro avg	0.48	0.45	0.44	118979
weighted avg	0.74	0.73	0.72	118979

```
Classification accuracy : 0.9869
Classification error : 0.0131
Precision (correctly predicted pos outcomes / all positive outcomes): 0.4461
```

Interpolated dataset - {rbf, c=1, gamma=auto}

	precision	recall	f1-score	support
falling	0.25	0.15	0.19	2192
lying	0.86	0.85	0.86	37566
lying down	0.26	0.22	0.24	4528
on all fours	0.38	0.39	0.38	3259
sitting	0.88	0.87	0.88	19243
sitting down	0.21	0.10	0.14	1218
sitting on the ground	0.93	0.41	0.57	7849
standing up from lying	0.43	0.45	0.44	13677
standing up from sitting	0.14	0.02	0.03	1035
standing up from sitting on the ground	0.19	0.54	0.28	2034
walking	0.83	0.95	0.89	25397
accuracy			0.73	117998
macro avg	0.49	0.45	0.44	117998
weighted avg	0.74	0.73	0.73	117998

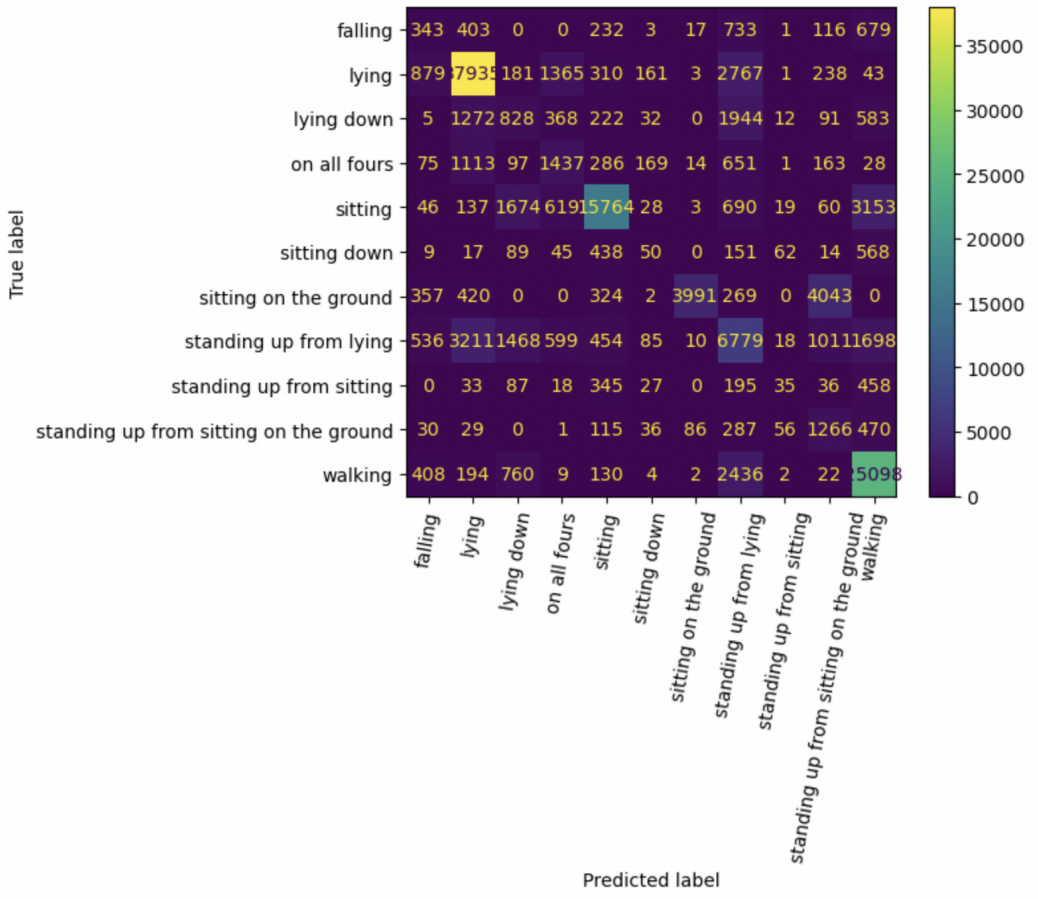
```
Classification accuracy : 0.9869
Classification error : 0.0131
Precision (correctly predicted pos outcomes / all positive outcomes): 0.4442
```

No Outliers dataset - {rbf, c=1, gamma=auto}

	precision	recall	f1-score	support
falling	0.15	0.01	0.02	2204
lying	0.78	0.91	0.84	37780
lying down	0.14	0.02	0.03	4605
on all fours	0.21	0.33	0.26	3374
sitting	0.70	0.75	0.72	19316
sitting down	0.00	0.00	0.00	1242
sitting on the ground	0.70	0.56	0.62	7849
standing up from lying	0.46	0.34	0.39	13934
standing up from sitting	1.00	0.00	0.00	1049
standing up from sitting on the ground	0.36	0.28	0.32	2038
walking	0.62	0.74	0.68	25588
accuracy			0.66	118979
macro avg	0.47	0.36	0.35	118979
weighted avg	0.63	0.66	0.63	118979

Classification accuracy : 0.9864
Classification error : 0.0136
Precision (correctly predicted pos outcomes / all positive outcomes): 0.0541

PCA dataset (PC1, PC2, PC3) - {rbf, c=1, gamma=auto}



SVC interpolated data confusion matrix (interpolated data)

Discussion

Unsupervised Learning

When the data without outliers was used in supervised learning, the model performance did not degrade, implying that the model reduced only the points that were redundant for our classification task. However, if we were to continue working with this project, it would be interesting to look more into the parameter selection of the Isolation Forest model to see if the results of our models could be improved, instead of being mostly unaffected. Testing the parameters in a grid search or a similar method could make the model more effective in not detecting any outliers falsely as normal points, or the other way around.

The idea of using PCA on our interpolated dataset was to see if the features could be reduced and still work just as good as predictors. Since there was no clear separation between the classes in the plotted PCs, PCA might not have been the best option for our dataset. The sensor coordinates for the ankles were highly correlated as well as the sensor coordinates for the belt and chest, which is noticeable from the direction of their vectors in the PC plots. Taking a mean of these features to get less correlated features before doing the PCA, might have resulted in some more interpretable results.

The z-coordinate is important for determining what activity is being carried out since it can classify them based on height, which is important in differentiating between lying, sitting or standing. Not scaling the data meant that the other coordinates with larger scale-values weighed heavier. Working more with scaling could make the coordinates equally weighted in the models. If we would continue working with this project, the possibility of doing other unsupervised methods would also be explored.

Supervised Learning

All KNN models had similar accuracy of around 0.75, it should be noted that the precision and recall varied greatly between different labels. The accuracy of the models did not change greatly with different K values, proving rather robust. Furthermore, all KNN models showed similar accuracy and had the same tendencies to misclassify some labels more often than others. Commonly occurring activities, such as lying and sitting were often correctly classified whereas less common activities such as standing up from sitting on the ground were more difficult to classify. This makes sense as the model has more data to rely on when guessing more common activities.

It appears that recall rate on all of the common activities was very high, the precision was however often somewhat lower. It is interesting to consider how the very high value that was selected based on accuracy affects this. It is likely that the accuracy rises because the common activities are guessed correctly more often. This might be at the expense of the less common activities losing recall. 130 and 200 nearest is a very high number considering that there were only around 150 data points with standing up from sitting in some of the test sets. If we were to continue this project, it might be of interest to study the accuracy reports and confusion matrix when searching for k-values.

It is interesting to see that the results were so similar between the different models. It was only during the final experiment, when many labels and data points were removed and new features, such as euclidean distance between sensors were used instead of the coordinates, that true change was observed. This model proved much better at predicting whether someone was lying. The accuracy of predicting sitting did however not improve as much. This might be because a lot of the sitting was happening in one part of the room, likely because a chair was placed there. Removing the coordinates of the sensors would hinder the models from using location to predict sitting.

The SVM model showed similar results as the KNN model. It had a nearly identical overall accuracy and made similar mistakes as the KNN model when comparing the confusion table. This might be due to the underlying similarity of the models, in spite of testing different kernels for the SVM model. Another reason could be that some activities in the dataset were simply more difficult to predict than others, as discussed above factors such as how common the activity was would effect how difficult they are to predict.

Overall, the models proved quite robust to changes in the dataset, such as including outliers, having only one sensor or using PC, as well as changes in the parameter K. It is very important to note that our model was probably not working in the way that we intended, as explained under the methods section. It was likely using room location to predict activity, which means that our models would be very poor at predicting activities in a new room or if someone were doing something unexpected, such as falling down in the middle of the corridor. One real world application of these models could be to identify when an elderly person has fallen. If the sensors only use room location to predict that, then the person would have to fall in the exact same spot every time. That is not very useful.
In order to predict transitional activities, such as standing up from lying or falling, one should explore new methods that involve time analyses and use information such as what activity the person was previously engaging in and rate of change. This could be an area for future studies.

References

Guido, Sarah, and Andreas C. Müller. 2016. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. N.p.: O'Reilly Media, Incorporated.

Kaluža, B., V. Mirchevska, E. Dovgan, M. Luštrek, and M. Gams. 2010. “An Agent Based Approach to Care in Independent Living.” 10.1007/978-3-642-16917-5_18.

Tokuç, Aylin. 2022. “Why Feature Scaling in SVM?” Baeldung. https://www.baeldung.com/cs/svm-feature-scaling.

UCI. 2010. “Localization Data for Person Activity Data Set.” UCI Machine Learning Repository. https://archive.ics.uci.edu/ml/datasets/Localization+Data+for+Person+Activity.

Liu, F. T., Ting, K. M., and Zhou, Z. H. 2008. Isolation forest. In Data Mining, 2008. ICDM’08. Eighth IEEE International Conference on (pp. 413–422). IEEE.