A data science approach to recognising activities from smartphone data

Technical Report

Isaac Tallack, u1615291

March 2019

Abstract

Human activity recognition (HAR) is a tool that has become increasingly common and easier to perform over the last few years with the huge growth of people using smartphones equipped with a wide range of different sensors and access to the internet. The uses for HAR are countless including healthcare, fitness and advertising. However, HAR algorithms are not always perfect and smartphone sensors produce a huge amount of data every second meaning there is a lot of interest around improving these classification algorithms not only for increased accuracy but also for speed and efficiency, particularly important for lower-power devices with small batteries like smartphones.

The aim of this report was to test and evaluate machine learning and data science methods to classify the activity in a given time window from sensor data generated from a smartphone (particularly accelerometer and gyroscope data). Different approaches to this problem are tested with the aim of maximising classification accuracy while keeping speed and efficiency in mind. The algorithms must be robust and generic enough classify unseen or unusual data as this will be vital in a real-world implementation.

The analysis found that very high performing classification algorithms could be created to not only classify activities but extended to detect falls with extremely high accuracy as well. A wide range of algorithms were tested but overall it seemed that the most information could be inferred from the frequency-domain representation of the accelerometer signals as these were the algorithms that performed best throughout. The main achievements of this report involved producing a fall detection classifier which performed 100% precision with minimal amounts of data in training/testing as well as an activity classifier that performed 99% precision classifying between five different activities.

Contents

1 Introduction

1.1 The Problem

Human Activity Recognition (HAR) is a useful tool in a wide range of sectors. From healthcare to advertising, the ability to classify how and in what way people are active can have great benefits.

Regarding healthcare, HAR has two clear uses: trip/fall detection and illness diagnosis. According to GOV.UK, “around a third of people aged 65 and over, and around half of people aged 80 and over, fall at least once a year” [1]. Trips/falls amongst elderly people can not only result in injuries, with potentially far-reaching consequences for future health and mobility, but also a loss of confidence or the need for physical supervision which would represent a loss of independence. Using HAR to detect trips/falls means that friends, family and emergency services can be alerted to an incident and action can be taken quickly without the need for physical supervision from carers or relatives which could undermine their sense of independence.

Furthermore, the ability for doctors to use HAR data from patients could be invaluable in identifying the cause of illnesses like heart disease, stroke, diabetes and depression and enable better advice to be given to patients as well as a way for patients to track their own progress and meet certain activity targets.

Finally, according to a 12-year study by the University of Cambridge, “eliminating inactivity in Europe would cut mortality rates by nearly 7.5%, or 676,000 deaths” [2]. One of the ways in which HAR could encourage activity is by providing a feedback mechanism to users (for example in fitness bands or mobile phones) which alerts them to periods of inactivity and motivates them to achieve certain fitness goals.

Regarding use for HAR in the private sector, particularly product development and advertising, information about user activity can be vital. Companies can get an insight into how people are using their physical products or mobile applications in order to achieve better usability and shape them to the way users interact with their product. What’s more, targeted advertising can benefit from HAR: for example, being able to identify a certain user does a lot of cycling and then target cycling products to them is far more attractive to advertisers than pushing ads to non-interested consumers.

1.2 Project Specification

The aim of this project is to produce a robust algorithm that can accurately classify human activities using data generated from a smartphone carried on the user’s body. The algorithm will do this by means of signal processing techniques combined with machine learning models to operate quickly and without the need of human assistance. Some examples of activities that could be classified include:

- Walking
- Running
- Cycling
- Sitting
- Standing
- Laying

The project will be predominately using accelerometer signals although further sensors may be added afterwards to see if it improves the accuracy of the algorithm. The scope of data being used for the project will be limited to public-domain datasets with adequate supporting literature to avoid the ethical concerns of collecting data from people. This will mean not only can the results be compared to results from prior literature in this area but also an ethical approval application is not needed so more time can be spent on the rest of the project.

1.3 Ethical Assessment

Ethics statement

I have read and completed the ethical flowchart questionnaire.

The outcome of the ethics statement is that ethics approval is NOT required. This is because my project does not involve:

- humans tissue/samples
- collecting new data involving people undertaking any task
- collecting data involving talking to people or taking information from people
- analysis of previously collected data that is not publicly available

Name: Isaac Tallack Date: 01/11/2018

2 Literature Review

2.1 Research papers

The UCI HAR Dataset (D. Anguita et al., 2013) [3] and MobiFall dataset (G. Vavoulas et al., 2013) [4] papers were both used throughout the experiments. Both describe the method in which data was collected along with different techniques used in the classification algorithms. Finally, the results from these classification algorithms are displayed which allow findings to be compared to other results gained from the same dataset and optimise the algorithms by basing them off a similar methodology.

One of the key pieces of literature used in research for the project was (Lara and Labrador, 2013) [5] which highlighted many challenges in producing human activity recognition (HAR) systems which gave a lot to think about in this project. The 7 main issues from the paper are summarised below:

- **Selection of attributes and sensors:** smartphones have a huge number of sensors (e.g. accelerometer, proximity, light, gyroscope, sound level) so there is a decision to be made about what sensors to use to achieve the best activity classification. The journal states that similar HAR algorithm have achieved classification in the high 90% region by using accelerometer data alone so this is where I will start my project.

- **Obtrusiveness:** having many sensors all over the subject will obviously increase activity classification accuracy but will also become an annoyance and intrude on the user. If the user cannot comfortably collect data for HAR then the system becomes redundant. As I am using smartphone data only this is not so much of a worry as people are used to carry a smartphone in their pocket.

- **Data collection protocol:** the journal talks about how testing the algorithm in a perfect lab environment compared to out in the real world can result in drastic drops in classification accuracy. Therefore, it is important to train the algorithm on a wide range of volunteers in realistic settings to make the model generic and robust.

- **Recognition performance:** there must be enough testing data that has been collected in a way similar to the training data to assess the performance of the model and standard machine learning metrics should be used.

- **Energy consumption:** there is a potential issue with energy consumption for classification depending on whether the classification is being on device (may take more processing power) or in the cloud (data/power needed to upload lots of data). In my experiment the classification will take place after recording and off the mobile device (on a PC) so energy consumption is not an issue.

- **Processing:** similar to the problem of energy consumption there are pros and cons to performing the activity classification on-or-off the device. As this project is a proof of concept this is again not an issue.

- **Flexibility:** training data needs to be generic and diverse so the model will be robust and adaptable to a wide range of use cases.

The paper also discusses different types of machine learning algorithms like decision trees and neural networks along with different approaches to the problem (supervised and unsupervised learning). Finally, evaluation metrics are discussed explaining the meaning of terms like accuracy, precision and recall.

A further journal (Wang et al., 2016) [6] talks about HAR using data collected from smartphone and focuses particularly on "sensor fusion": the combination of different signals, in particular accelerometer and gyroscope and looks at how the combination of these sensors gives superior results to a signal sensor approach. However, this also shows how the accelerometer data alone is far more useful than the gyroscope alone, the combined signal far outperforms the gyroscope-only signal (+30%) whereas marginally improves the accelerometer signal (+3%).

Finally, (Davis et al., 2016) [7] has its focus on improving quality of life for the elderly, a key driving force for this project, in reducing social/health care. A number of models are evaluated, and the best model returns an impressive classification accuracy of 99.7%.

2.2 Fourier transforms/signal processing techniques

The purpose of the Fourier transform is to decompose a signal that is a function of time into the frequencies that make up the signal. This is a useful signal processing technique for this project as the smartphone data from different activities may have dominant frequencies. For example, a report about the frequency of people walking found "that on shopping floors the people walk with an average frequency of 2.0Hz... but on the footbridges they walk with an average frequency of 1.8Hz" [8]. This clearly shows that there is valuable information to be found in the frequency domain of the signals.

The `fft` function in the `scipy.fftpack` Python package takes an n-length real or complex array in the form $[x_0, x_1, ..., x_{n-1}]$, performs the discrete Fourier transform and returns a complex array in the form $[y_0, y_1, ..., y_{n-1}]$. Each element in the output array is calculated following the formula shown in equation 1 [9]

$$y(j) = \sum_{k=0}^{n-1} x_k \cdot e^{-\frac{i2\pi}{n}kj}, \;\; j \in [0,\; n-1] \tag{1}$$

where j is the frequency.

Term-by-term this equation computes the amplitudes of different frequency components in the signal.

Figure 1 shows an example FFT (in Python) with a simple signal with two known frequency components. This signal was run through the Fourier transform algorithm and the result was plotted. The top of figure 1 shows the time-domain representation of the curve $cos(2\pi f_1 t + 2\pi f_2 t)$ where f_1 and f_2 are 2Hz and 5Hz respectively. At the bottom of figure 1 is the frequency-domain representation of the same curve. The two peaks in the frequency-domain graph correctly appear at 2Hz and 5Hz.

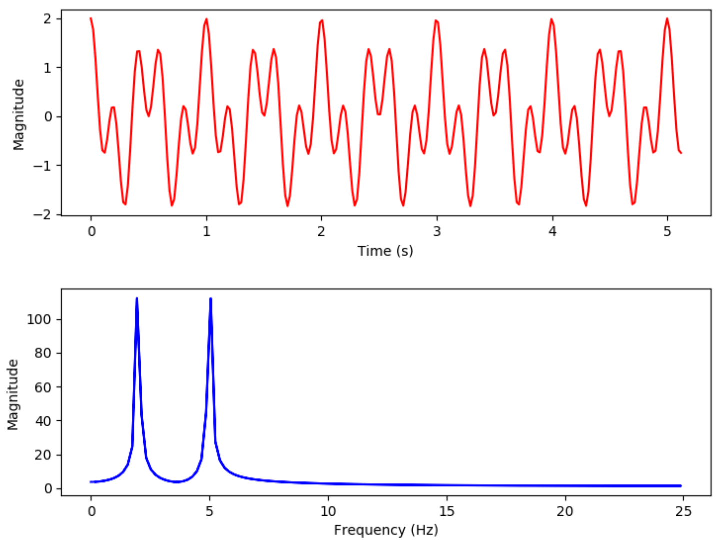


Figure 1: Fourier transform of a simple two-frequency cosine curve

2.3 Machine learning algorithms

The performance of three different machine learning algorithms will be tested for their ability to classify different activities through data produced by smartphone sensors. The quality of performance will be assessed using the evaluation metrics discussed in section 2.4.

2.3.1 Decision Tree (DT)

The first and perhaps the simplest machine learning algorithm used is the decision tree (DT). A decision tree classifier learns by constructing simple if-then statements to represent patterns that occur in the data. Each if-then statement is called a fork and will separate into two different branches each with further if-then statements or alternatively a classification label/outcome (known as a leaf node).

For example, in figure 2 we are trying to classify if a person is fit or unfit. The first branch asks whether the person is younger than 30 years old. Based on the answer to the first question we move onto one of two new questions: whether the person eats lots of pizza or whether the person exercises in the morning. Finally, based on the answer to that final question we reach a leaf node which determines whether the given person is fit or unfit. A visual example produced by *R2D3* explains this algorithm in a very intuitive and interactive way and was very helpful in my research [10].

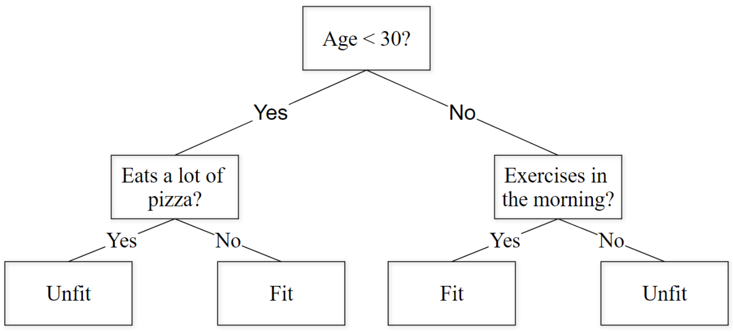


Figure 2: Example of simple decision tree to determine whether a person is fit or unfit, adapted from [11]

The decision tree algorithm has the advantage of being very easy for humans to understand and interpret and has very quick classification and learning times as it is only simple if-then logic statements. However, a disadvantage of this type of learning is that it is prone to overfitting the data: this results in very large and complex trees that are not well generalised to classify new pieces of data.

2.3.2 *k*-nearest neighbours (*k*NN)

The next algorithm to be used in the experiments is the k-nearest neighbours algorithm (k-NN). This is again a very simple machine learning algorithm and is easy to visualise and understand. Each piece of training data that the algorithm is trained on is a multi-dimensional vector with a corresponding label. New pieces of data are then classified and labelled based on the label of the k pieces of data that fall closest in distance to the new data. This is better visualised by figure 3.

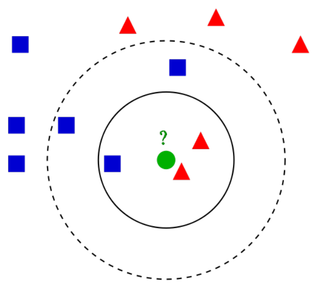


Figure 3: Visual representation of k-NN algorithm [12]

As can be seen in the figure, the red triangles and blue squares are pieces of training data with two different labels that have been plotted based on the values in their features vectors. The green circle is a new piece of data that needs to be classified. The solid black circle shows the result of classification if our k value is set to 3 (meaning we are looking for the 3 closest pieces of training data to determine the label of our new data). If this is the case, our unknown data point is classified as a red triangle as the majority of training points inside our ring are red triangles. However, if k is set to 5, as shown by the dashed circle then the unknown point will be classified as a blue square because those training data pieces now appears more times.

An obvious advantage to this algorithm is how easy it is to implement and understand how it is working. However, disadvantages to this algorithm include the fact that the training data is only used during classification - distances are computed at classification which can make this algorithm slow if there is lots of training data with many features. Furthermore, an optimal value of k has to be determined manually for best results - a change in k can change the predicted label.

2.3.3 Support Vector Machine (SVM)

The final algorithm to be used in the experiments is the support vector machine (SVM). This is the most complex algorithm to understand in terms of inner workings although will be easy to implement and assess by using the `scikit-learn` package in Python. The aim of the SVM is to construct an optimal hyperplane through the training data points such that the hyperplane has the greatest distance from the nearest point in either class. Figure 4 shows a visual example of a linear hyperplane separating two classes of data.

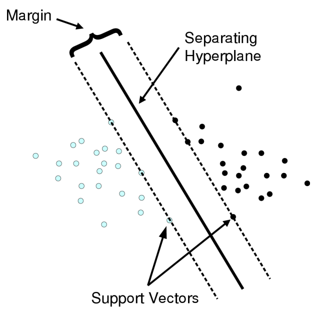


Figure 4: SVM hyperplane optimisation [13]

As can be seen in the diagram, the position of the plane is optimised such that the margin is as large as possible - the closest data points are as far from the line as possible. The SVM algorithm works on high length feature vectors and will construct a more complex multi-dimensional hyperplane to cope with this. In SVM learning there is a choice of kernel, the kernel in figure 4 is a linear kernel (as it is a straight line that splits the data) however in some instances a polynomial hyperplane may produce better results. In addition to the kernel type there are many other parameters in the SVM classifier that can be tuned in order to produce better classification results.

The advantages of the SVM algorithm is that it works well on high-dimensional data and that it is well generalised so there is a reduced chance of overfitting to the data. However, the disadvantages of the SVM algorithm is that it can be hard to choose the right kernel and parameters and it is difficult to interpret the final model (unlike the decision tree and kNN).

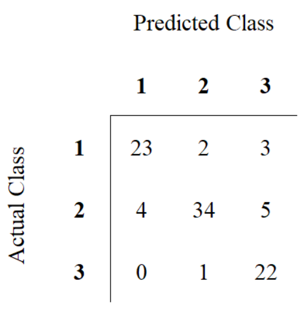
2.4 Machine learning model evaluation metrics

Two functions from the `sklearn.metrics` Python package will be used to assess how well the classification models are working: the confusion matrix and the classification report. The classification report will give a good overview high-level of how well our algorithm is working whereas the confusion matrix will show in more depth the strengths and weaknesses of our model.

2.4.1 Confusion matrix

The confusion matrix is a detailed representation of how different labels are being predicted. The data is presented in an x-by-x table where x represents the number of different classes in the data labels.

Figure 5 shows an example of what a confusion matrix looks like. The confusion matrix shown is classifying the data into three labels. Looking horizontally tells us about how many data pieces were actually labelled in a given class whereas looking vertically tells us how the data was predicted in each class. For example, 23 pieces of data were correctly classified in label 1. However, 2 pieces of data were labelled as class 1 but incorrectly predicted as being from class 2. A perfect classification algorithm would yield all the number in a diagonal line with zeros in all the other boxes.



		Predicted Class		
		1	**2**	**3**
Actual Class	**1**	23	2	3
	2	4	34	5
	3	0	1	22

Figure 5: Example confusion matrix

This visual representation of the classification is extremely powerful in allowing us to understand which classes are being misclassified most often and which classes are being mixed up most often, e.g. are most alike in features.

2.4.2 Classification report

The classification report will give us a broad overview of how well our classifier is performing. It will do this by returning precision, recall, f1-score and support values for each of the labels in our dataset along with an overall average [14].

- The **precision** tells us what percentage of the instances classified as one label were correctly classified. This is calculated as the ratio of true positives to the sum of true and false positives.
- The **recall** describes how many instances that were in a given class were correctly classified. This is calculated as the ratio of true positives to the sum of true positives and false negatives.
- The **f1-score** is a weighted average of precision and recall. An f1-score of 1.0 is the best and 0.0 is the worst.
- The **support** tells us how many times each class appeared in the dataset. This helps to tell us if there is an imbalance in our classes which could mean worse classification precision for a certain activity.

3 Method

The methodology is split into three main sections: data collection/sourcing, signal processing and machine learning/classification. The steps taken in each part of the project pipeline are outlined in the relevant sections below. A visual representation of the project pipeline can be seen in Figure 6.

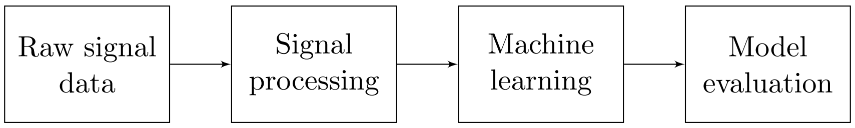


Figure 6: Flowchart outlining project pipeline

3.1 Data collection/sourcing

This part of the project involved sourcing relevant datasets that could be used in activity classification algorithms. The key features of a dataset that would make it suitable for this project were:

- Data recorded from a mobile phone on the subject
- A wide range of activities performed
- Clear activity labelling
- Clear dataset methodology
- Useful data (e.g. accelerometer and gyroscope)

3.1.1 UCI HAR Dataset

The first dataset used for this project was the 'Human Activity Recognition Using Smartphones Data Set' from the UCI Machine Learning Repository [3]. This dataset has accelerometer and gyroscope readings from a waist-mounted smartphone for various volunteer subjects performing six different activities: walking, walking upstairs, walking downstairs, sitting, standing and laying.

The dataset contains 10,299 activity windows, randomly split such that 70% is for training a classification algorithm and the other 30% for testing it. The activity windows are clearly labelled 1 to 6 where a '1' represents walking, a '2' walking upstairs and so on. Each window is 2.56 seconds in length, resulting in 128 readings at a sampling rate of 50Hz. For this experiment the 'body' acceleration values were used - this means the acceleration due to gravity was removed.

Little processing was required to make the data useable for the classification algorithm. Using the pre-defined 2.56-second windows for classification, the `genfromtxt` function was used to create `numpy` arrays from the data within Python as shown in code listing 1.

Listing 1: Pre-processing of UCI HAR dataset

```
import numpy as np

x_values = np.genfromtxt("...UCI␣HAR␣Dataset\\train\\
   inertial_signals\\body_acc_x_train.txt", delimiter = "\n")
y_values = np.genfromtxt("...UCI␣HAR␣Dataset\\train\\
   inertial_signals\\body_acc_y_train.txt", delimiter = "\n")
```

```
z_values = np.genfromtxt("...UCI␣HAR␣Dataset\\train\\
   inertial_signals\\body_acc_z_train.txt", delimiter = "\n")
labels = np.genfromtxt("...UCI␣HAR␣Dataset\\train\\y_train.csv",
   delimiter = ",")
```

3.1.2 MobiFall and MobiAct

The second dataset found for this project was the second release of the MobiFall dataset [4]. This dataset includes data recorded from a single smartphone in the pocket of the varying subjects while numerous activities were performed. These include the same walking, walking upstairs, walking downstairs, sitting, standing and laying activities as seen in the UCI HAR dataset amongst a few other new activities including four types of falls.

In this dataset, trials from each activity are stored in their own specific folder so labelling of data is obvious. Preparing this dataset for use in classification was slightly more difficult as the length of activities were not uniform (e.g. walking activities were 5 minutes in length whereas walking up/down stairs were 10 seconds). To work around this Python was used to write a function which takes a directory, activity label and an optional parameter to split the data x times. As these windows weren't always exactly the length stated in the `readme.txt` the shortest window would be recorded and later all other windows would be shortened down to this lowest common denominator. The code for this can be seen in code listing 2.

Listing 2: Pre-processing of MobiFall dataset

```
import numpy as np

np_acc = []
labels = []
shortest_element = 1000

def gen_test_train(directory, label, split = 1, skip_header = 16)
   :
    global shortest_element
    for file in os.listdir(directory):
        if (file[4:7] == "acc" and file.endswith(".txt")):
            l_window = np.genfromtxt(os.path.join(directory, file
               ), delimiter = ',', skip_header=skip_header,
               usecols=(1, 2, 3))
            s_window = np.array_split(l_window, split)
            for window in s_window:
                if len(window) < shortest_element:
                    shortest_element = len(window)
                np_acc.append(window)
                labels.append(label)
```

An example call to this function using data from sitting activities (originally 1 minute in length but being split to 10 seconds) is shown in code listing 3.

Listing 3: Pre-processing of MobiFall dataset

```
SIT = "...MobiAct_Dataset_v2.0\\Raw␣Data\\SIT"

gen_test_train(SIT, 1, split = 6)
```

The output array `np_acc` is a two-dimensional array of 10-second windows and the `labels` array is a one-dimensional array of activity labels.

3.2 Signal processing

Signal processing is an important middle-step between getting the data in a useable format and feeding data into a machine learning algorithm. By performing processing on the data we can extract new features from the data which weren't immediately apparent from the raw numbers.

The first example of signal processing performed on the data on was the Fourier transform (discussed in literature review, section 2.2). By performing a Fourier transform on the data we can see the spectrum of dominant frequencies present in the time-domain representation of the data. To perform the Fourier transform on the data the `fftpack` in the `scipy` library for Python was used as it has good documentation and is easy to use. The code for the Fourier transform can be seen in code listing 4. It takes a time-domain data array as well as the sampling time period as inputs and returns the Fourier transform of the array as well as the array of Fourier transform sampling frequencies.

Listing 4: Fourier transform of time-domain data

```
import scipy as sy
import scipy.fftpack as syfp
import numpy as np

def fourier_transform(array, t_period):
    length = len(array)
    x = sy.linspace(t_period, length*t_period, num=length)

    f = syfp.fft(array)
    yf = syfp.fftfreq(length, np.mean(np.diff(x)))

    return f, yf
```

The second part of the signal processing was feature extraction. The aim of this was to produce a function that would take an array of time-domain values as an input and would return an array of features extracted from the data. The following features were extracted for training the algorithm:

Time-domain:

- Maximum
- Minimum
- Mean
- Standard deviation
- Median
- Average sum-of-squares
- Interquartile range

Frequency-domain:

- Maximum magnitude
- Frequency at maximum magnitude
- Mean
- Standard deviation
- Median
- Skew
- Kurtosis
- Interquartile range
- Energy

The code in listing 5 shows the function for the feature extraction. The function takes a time-domain value input and returns a one-dimensional list of features.

Listing 5: Feature extraction function

```
def comp_ftrs(t_values):
    fft_values = syfp.fft(t_values)
    length = len(t_values)
    x = sy.linspace(0, length*0.005, num=length)
    f = syfp.fftfreq(length, np.mean(np.diff(x)))

    MSMMM = [np.mean(t_values), np.std(t_values), np.median(
       t_values), np.amax(t_values), np.amin(t_values)]
    ASoS = [(sum(val*val for val in t_values))/128]
    IQR = [np.subtract(*np.percentile(t_values, [75, 25]))]
    f_max_mag = [np.amax(fft_values)]
    f_max_freq = [f[list(fft_values).index(f_max_mag)]]
    f_MSM = [np.mean(fft_values), np.std(fft_values), np.median(
       fft_values)]
    f_skew = [sy.stats.skew(fft_values)]
    f_kurtosis = [sy.stats.kurtosis(fft_values)]
    f_IQR = [np.subtract(*np.percentile(fft_values, [75, 25]))]
    Energy = [(sum(f_val*f_val for f_val in fft_values))/128]
    return MSMMM+ASoS+IQR+f_max_mag+f_max_freq+f_skew+f_kurtosis+
       f_IQR+f_MSM+Energy
```

3.3 Machine Learning

Now that the data has had useful features extracted it is time to prepare the data for the machine learning phase. In this phase the data will be split into training and testing sets, a number of different ML (machine learning) algorithms will be trained on the training data and then the model will be evaluated on its success at classifying the new, unseen testing data.

Since the UCI HAR dataset has training and testing data pre-separated but the MobiFall dataset does not, the training and testing data will be separated manually using `train_test_split` from the `sklearn` package in Python. This way the algorithm will work on both datasets natively without needing to change approach.

Following the data separation, a classifier of choice (either a *k*NN, SVM or Decision Tree) will be trained on the training data. The classifier will then be instructed to make predictions on the label of new pieces of data from the testing section. After this is completed, a confusion matrix and classification report (discussed in the literature review) will be printed to assess how well the model has performed. An example of this process using an SVM classifier can be seen in code listing 6.

Listing 6: Machine learning classification

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
   confusion_matrix
from sklearn import svm

X_train, X_test, y_train, y_test = train_test_split(data, labels,
    test_size=0.20)

classifier = svm.SVC()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

4 Results

4.1 UCI HAR Dataset

4.1.1 Dominant frequency and magnitude training (experiment 1)

The first tests were done on activity classification. To begin with, the problem was boiled down to something very simple: how accurately an activity could be predicted in a kNN classifier using values for only the dominant frequency in the accelerometer signal (a spike in the Fourier transform, see section 2.2) and corresponding magnitude at the dominant frequency. What's more, to begin with the classifier will only train and test using the accelerometer signal in one-dimension (the x-axis). This makes the problem simpler as we are only using two values per activity window for training and classification.

The classification report from this test is displayed in table 1.

Table 1: Classification report for x-axis kNN classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.70	0.83	0.76	254
2. Walking upstairs	0.61	0.54	0.57	208
3. Walking downstairs	0.68	0.63	0.65	187
4. Sitting	0.39	0.45	0.42	261
5. Standing	0.43	0.44	0.43	259
6. Laying	0.63	0.53	0.58	302
Average/total	0.57	0.56	0.56	1471

The next obvious step was to add in the same values (dominant/maximum frequency and corresponding magnitude) from the other y-axis and z-axis to the training algorithm and investigate whether the other axes add value to classification. This would mean six features per activity window in training and testing. The classification report from this test is displayed in table 2. The confusion matrix from this same test can be seen in table 3. The numbers for each activity in the confusion matrix correspond to those seen in the classification report prior.

Table 2: Classification report for all-axes kNN classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.75	0.85	0.80	232
2. Walking upstairs	0.70	0.79	0.74	202
3. Walking downstairs	0.92	0.71	0.80	222
4. Sitting	0.48	0.49	0.48	286
5. Standing	0.52	0.61	0.56	251
6. Laying	0.81	0.68	0.74	278
Average/total	0.69	0.67	0.68	1471

Table 3: Confusion matrix for all-axes kNN classification

	1	**2**	**3**	**4**	**5**	**6**
1	197	30	5	0	0	0
2	34	159	9	0	0	0
3	28	37	157	0	0	0
4	1	1	0	139	114	31
5	0	0	0	86	152	13
6	2	0	0	63	25	188

Now this had returned a fairly high average precision it seemed like the next logical step was to train and test an SVM classifier on the exact same train/test split to see which model produces better precision. The classification report and confusion matrix for this experiment and be seen in tables 4 and 5 respectively.

Table 4: Classification report for all-axes SVM classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.81	0.86	0.83	232
2. Walking upstairs	0.75	0.80	0.78	202
3. Walking downstairs	0.83	0.77	0.80	222
4. Sitting	0.53	0.49	0.50	286
5. Standing	0.55	0.61	0.61	251
6. Laying	0.78	0.68	0.72	278
Average/total	0.70	0.67	0.70	1471

Table 5: Confusion matrix for all-axes SVM classification

	1	**2**	**3**	**4**	**5**	**6**
1	199	23	10	0	0	0
2	24	162	16	0	0	0
3	22	30	170	0	0	0
4	0	1	2	133	110	40
5	0	0	0	64	174	13
6	1	0	6	52	31	188

4.1.2 Frequency-domain 'binned' training (experiment 2)

In these experiments only the frequency-domain accelerometer signals will be used in classification. The way this will be done is by feeding the machine learning algorithm magnitude values at x evenly spaced intervals throughout the signal (bins). These binned values will be taken for all axes of the accelerometer. The effect of changing the number of bins will be assessed, x, as well as the machine learning model affects the precision and how this time-domain approach compares to the frequency-domain one used in section 4.1.1.

Table 6 shows how changing the number of bins affects the precision of the classification along with the time it takes to train and classify.

Table 6: Classification report averages for all-axes binned kNN classification

	Average/total				**Time taken (s)**
Number of bins	**Precision**	**Recall**	**F1-score**	**Support**	
10	0.62	0.60	0.60	1471	5.79
25	0.74	0.72	0.72	1471	8.13
50	0.77	0.76	0.76	1471	12.10
80	0.81	0.80	0.80	1471	16.85
128	0.83	0.81	0.82	1471	25.21

A more in-depth look at the best performing kNN binned classification; the full 128-value time-domain model can be seen in the classification report (table 7) and confusion matrix (table 8).

Table 7: Classification report for all-axes 128-bin kNN classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.97	1.00	0.99	226
2. Walking upstairs	0.98	0.98	0.98	246
3. Walking downstairs	1.00	0.95	0.98	176
4. Sitting	0.52	0.65	0.58	255
5. Standing	0.69	0.65	0.67	292
6. Laying	0.91	0.74	0.81	276
Average/total	0.83	0.81	0.82	1471

Table 8: Confusion matrix for all-axes 128-bin kNN classification

	1	**2**	**3**	**4**	**5**	**6**
1	226	0	0	0	0	0
2	4	242	0	0	0	0
3	2	6	168	0	0	0
4	0	0	0	166	74	15
5	0	0	0	95	191	6
6	0	0	0	59	13	204

The same experiment but using a SVM classifier instead of the kNN one used prior was then performed. Table 9 shows how changing the number of bins affects the precision of the classification along with the training and testing time for the SVM classifier.

Table 9: Classification report averages for all-axes binned SVM classification

	Average/total				**Time taken (s)**
Number of bins	**Precision**	**Recall**	**F1-score**	**Support**	
10	0.60	0.61	0.58	1471	7.38
25	0.70	0.71	0.66	1471	10.85
50	0.71	0.72	0.68	1471	16.65
80	0.77	0.74	0.71	1471	23.85
128	0.79	0.71	0.71	1471	36.96

A more in-depth look at the best performing SVM binned classification; the full 128-value time-domain model can be seen in the classification report (table 10) and confusion matrix (table

11).

Table 10: Classification report for all-axes 128-bin SVM classification

Activity	Precision	Recall	F1-score	Support
1. Walking	1.00	0.99	1.00	237
2. Walking upstairs	0.99	0.99	0.99	216
3. Walking downstairs	0.97	1.00	0.98	183
4. Sitting	0.35	0.78	0.49	251
5. Standing	0.58	0.26	0.36	278
6. Laying	0.92	0.46	0.61	306
Average/total	0.79	0.71	0.71	1471

Table 11: Confusion matrix for all-axes 128-bin SVM classification

	1	**2**	**3**	**4**	**5**	**6**
1	235	2	0	0	0	0
2	0	242	3	0	0	0
3	0	0	183	0	0	0
4	0	0	0	196	47	8
5	0	0	0	201	73	4
6	0	0	3	158	5	140

4.1.3 Feature-based training (experiment 3)

The next experiment involved training the machine learning algorithms on a feature vector generated from the time-domain and frequency-domain signals. This is similar to the method used in the research paper relating to the dataset [3]. The approach for this test is discussed further in section 3.2. Each activity window will be summarised by a 48-value feature vector made up of the following values in all 3-axes:

Time-domain:

- Maximum
- Minimum
- Mean
- Standard deviation
- Median
- Average sum-of-squares
- Interquartile range

Frequency-domain:

- Maximum magnitude
- Frequency at maximum magnitude
- Mean
- Standard deviation
- Median
- Skew
- Kurtosis
- Interquartile range
- Energy

This feature vector is then fed to the machine learning algorithm (either a kNN, SVM or DT) with a label for learning. New windows are classified based on their feature vectors.

Results for the kNN, accelerometer-only, feature-based training can been seen in table 12.

Table 12: Classification report for all-axes accelerometer feature-based kNN classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.78	0.93	0.85	243
2. Walking upstairs	0.84	0.73	0.78	214
3. Walking downstairs	0.94	0.86	0.90	200
4. Sitting	0.37	0.38	0.37	251
5. Standing	0.55	0.56	0.55	287
6. Laying	0.70	0.66	0.68	276
Average/total	0.68	0.68	0.68	1471

The results from the same feature-based tests run on an SVM classifier can be seen in table 13.

Table 13: Classification report for all-axes accelerometer feature-based SVM classification

Activity	Precision	Recall	F1-score	Support
1. Walking	0.85	0.91	0.88	243
2. Walking upstairs	0.86	0.81	0.83	214
3. Walking downstairs	0.93	0.88	0.91	200
4. Sitting	0.43	0.29	0.34	251
5. Standing	0.59	0.69	0.64	287
6. Laying	0.66	0.76	0.70	276
Average/total	0.70	0.71	0.70	1471

4.1.4 Addition of gyroscope data (experiment 4)

Table 14 shows how the addition of gyroscope data affects the precision score of each of the experiments described prior. For each experiment the same test was run using only-accelerometer data, only-gyroscope data and then both accelerometer and gyroscope data (combination). The gyroscope signal was subject to identical data processing, for example: in experiment 1 a Fourier transform of the signal will be performed and the maximum frequency and magnitude selected for both sensors.

Note: in experiment 2 the 128-bin data will be used as it returned the highest precision score.

Table 14: Precision of classification comparision for different sensor types

	Accelerometer		**Gyroscope**		**Combination**	
Experiment	***k*NN**	**SVM**	***k*NN**	**SVM**	***k*NN**	**SVM**
1	0.69	0.70	0.65	0.65	0.74	0.76
2	0.83	0.79	0.85	0.76	0.88	0.82
3	0.68	0.70	0.65	0.61	0.71	0.68

4.1.5 Summary

The graph in figure 7 shows a comparison between the precision achieved by the two different classifiers (kNN and SVM) for the three experiments covered in section 4.1 on accelerometer-only signal data.

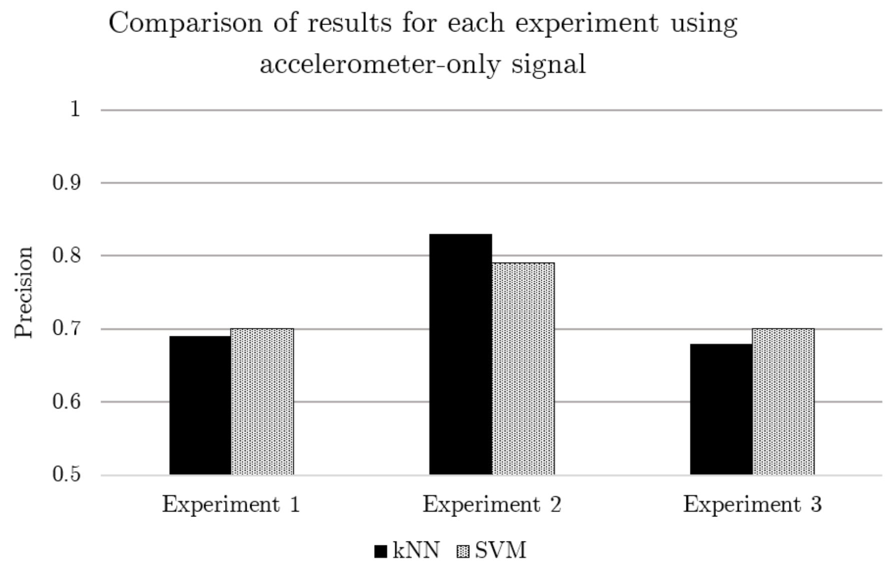


Figure 7: Graph to compare precision classification for all experiments for both classifier types using accelerometer-only signals

The graph in figure 8 shows a comparison between the precision achieved by the best performing algorithm (experiment 2, accelerometer and gyroscope signals, kNN classifier) and the journal (D. Anguita et al., 2013) [3] on an activity by activity basis.

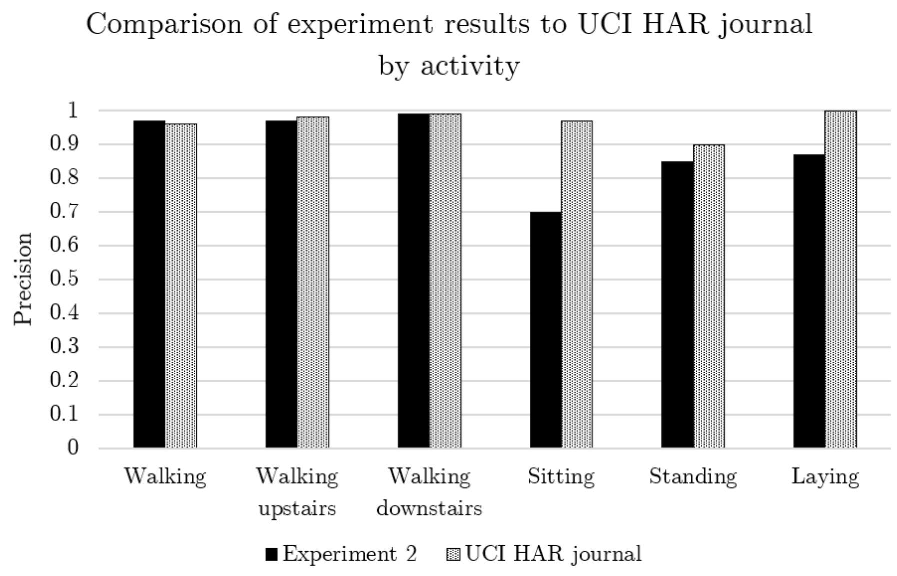


Figure 8: Graph to compare precision classification performance by activity from experiment 2 to those in the UCI HAR journal

4.2 MobiFall and MobiAct

The MobiFall/MobiAct dataset contains 16 different activities: 4 of which are falls and the rest daily activities. Each activity is given a three-letter code for identification and labelling. Not all the activities available in the dataset were used in my experiment so the activity codes of the relevant activities can be seen in tables 15 and 16 [4].

Table 15: Activity codes and corresponding name/description for "fall" activities in dataset

Activity code	**Name**	**Description**
FOL	Forward-laying	Fall forward from standing onto hands to dampen fall
FKL	Front-knees-lying	Fall forward from standing onto knees to dampen fall
SDL	Sideward-lying	Fall sideways from standing bending knees
BSC	Back-sitting-chair	Fall backwards trying to sit on chair

Table 16: Activity codes and corresponding name/description for "non-fall" activities in dataset

Activity code	**Name**	**Description**
STD	Standing	Standing with subtle movements
WAL	Walking	Normal walking
STU	Walking upstairs	Stairs up (10 stairs)
STN	Walking downstairs	Stairs down (10 stairs)
SIT	Sitting on chair	Sitting on a chair with subtle movements

4.2.1 Fall Detection

The first part of this experiment involved classifying activity windows into falls and non-falls. The fall detection algorithm is designed to classify FOL, FKL, SDL and BSC against STD and SIT activities.

To begin with, all the time-domain accelerometer values were fed into an SVM classifier with fall activities (FOL, FKL, SDL, BSC) being labelled a '1' (fall) and non-fall activities (STD, SIT) a '0' (no fall). The classifier was then tasked with predicting labels for the other 20% of the data (partitioned exclusively for testing). The classification report and confusion matrix of results is shown in table 17 and 18 respectively.

Table 17: Classification report for standing falls using all time-domain data and an SVM classifier

Activity	Precision	Recall	F1-score	Support
Non-fall	1.00	1.00	1.00	387
Fall	1.00	1.00	1.00	150
Average/total	1.00	1.00	1.00	537

Table 18: Confusion matrix for standing falls using all time-domain data and an SVM classifier

	0	**1**
0	387	0
1	0	150

This algorithm is perfectly classifying falls against non-fall activity windows although requires using **4,681,026** data points to do so.

Precision of classification was then tested feeding various classification algorithms (see 2.3) different feature vectors with the aim of reducing processing power and speeding up the classification. Incorporating the feature extraction algorithm used in section 4.1.3, the classifier could be trained on different combinations of features. Table 19 shows average precision for different feature combinations along with type of classifier, time for classification and number of data points required in training. This process was trial and error adding and removing features.

Table 19: Comparison of precision and efficiency in fall detection for varying training data

Training data	**Precision**	**No. of data points**	**Classifier**	**Time taken (s)**
All time-domain values	1.00	4,681,026	SVM	5.766
Mean, median and standard deviation	1.00	24,129	SVM	0.022
Frequency-domain dominant frequency and max magnitude	1.00	16,086	kNN	0.008
Standard deviation	1.00	8,043	kNN	0.007
Frequency-domain dominant frequency and max magnitude	0.99	16,086	DT	0.008
Standard deviation	1.00	8,043	DT	0.006

4.2.2 Activity Classification

Note: there was a problem with the SVM classifier for these tests so only kNN and DT algorithms will be used. This problem is discussed in section 5.2.2.

The next part of the experimentation using the MobiFall dataset was activity classification similar to that performed throughout section 4.1. The activities to be classified in this section are as follows: walking (WAL), walking upstairs (STU), walking downstairs (STN), sitting (SIT) and standing (STD).

Firstly, following an approach similar to that in section 4.1.1 (dominant frequency and magnitude) trained on a kNN classifier yielded results shown in table 20.

Table 20: Classification report for dominant frequency and magnitude using kNN classifier

Activity	Precision	Recall	F1-score	Support
1. Walking	0.85	0.83	0.84	339
2. Walking upstairs	0.67	0.49	0.57	85
3. Walking downstairs	0.60	0.27	0.38	91
4. Sitting	0.95	0.95	0.95	22
5. Standing	0.81	0.98	0.89	358
Average/total	0.79	0.81	0.79	895

Secondly, following an approach similar to that in section 4.1.2 (binned frequency-domain values) trained on a kNN classifier yielded results shown in table 21.

Table 21: Classification report for binned frequency-domain using kNN classifier

Activity	Precision	Recall	F1-score	Support
1. Walking	0.99	0.99	0.99	359
2. Walking upstairs	0.97	0.91	0.94	77
3. Walking downstairs	0.91	0.96	0.94	75
4. Sitting	1.00	1.00	1.00	29
5. Standing	1.00	1.00	1.00	355
Average/total	0.99	0.99	0.99	895

The confusion matrix for this experiment can be seen in table 22.

Table 22: Confusion matrix for binned frequency-domain using kNN classifier

	1	**2**	**3**	**4**	**5**
1	349	0	1	0	0
2	1	63	5	0	0
3	2	0	74	0	0
4	0	0	0	22	1
5	0	0	0	0	377

Thirdly, following an approach similar to that in section 4.1.3 (signal feature vector) trained on a kNN classifier yielded results shown in table 23.

Table 23: Classification report for feature vector approach using kNN classifier

Activity	Precision	Recall	F1-score	Support
1. Walking	0.74	0.87	0.81	365
2. Walking upstairs	0.38	0.30	0.34	69
3. Walking downstairs	0.38	0.12	0.19	88
4. Sitting	0.95	0.91	0.93	23
5. Standing	0.87	0.93	0.90	350
Average/total	0.74	0.77	0.75	895

4.2.3 Summary

Figure 9 shows a graph comparing activity classification performance for the three experiments performed in this section using both a kNN and DT classifier.

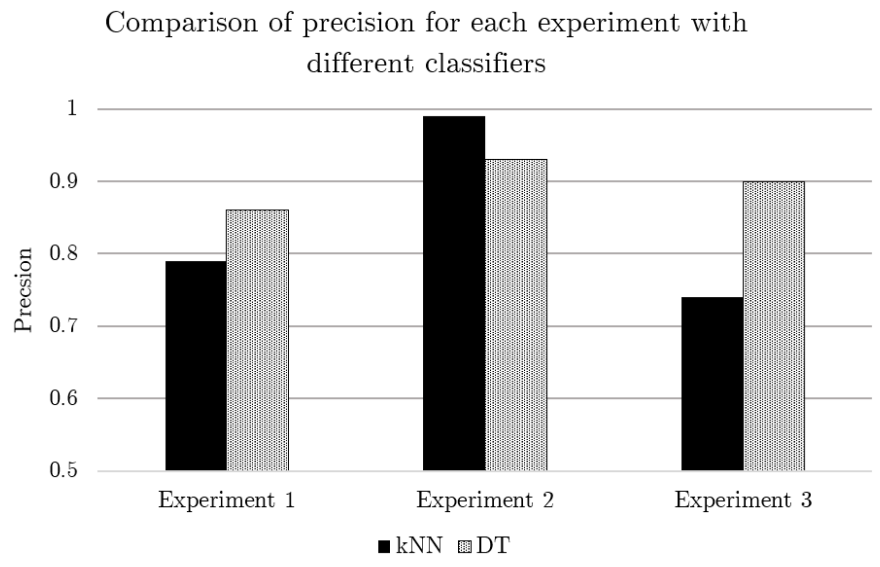


Figure 9: Graph to compare precision classification for all experiments for both classifier types

Figure 10 shows a graph comparing activity classification performance between data from the MobiFall dataset and data from the UCI HAR dataset. Classification precision is taken from the best two performing algorithms in each respective dataset: both happen to be the fully-binned frequency-domain data trained on a kNN classifier.
Note: the 'laying' activity was not part of my training set for the MobiFall dataset so classification precision for this activity is not compared.

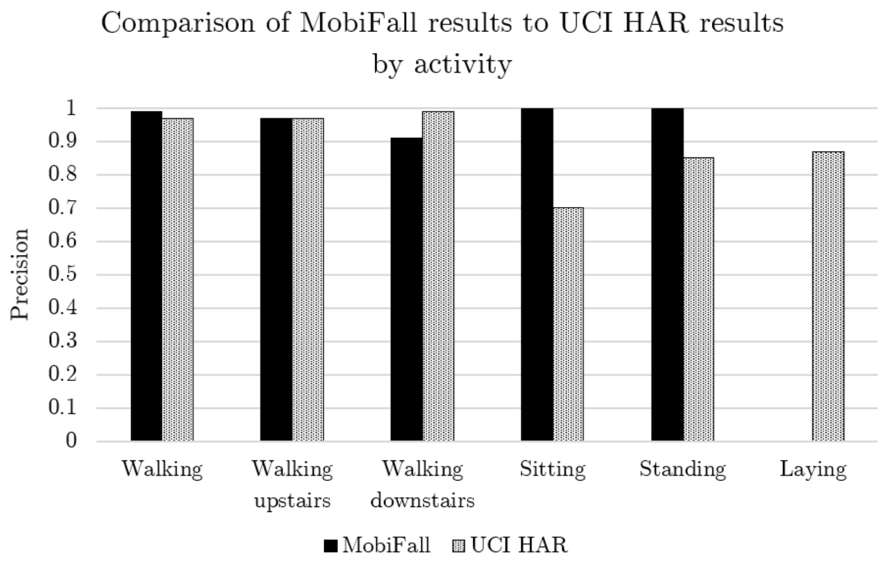


Figure 10: Graph to compare precision classification for MobiFall dataset against the UCI HAR dataset by activity

5 Analysis

5.1 UCI HAR Dataset

5.1.1 Dominant frequency and magnitude training (experiment 1)

To randomly guess labels from the data without any prior learning we would expect precision of around 17% so even this very simple model has produced a passable result.

To check the results of the single-axis dominant frequency approach in a visual way, histograms were plotted for the features the model was trained on in order to see the differences between the activities. On the x-axis is the maximum/dominant frequency and on the y-axis is the magnitude of that maximum/dominant frequency. Darker areas indicate more points falling in that region whereas whiter areas show a lack of points in that region. These plots can be seen in figure 11.

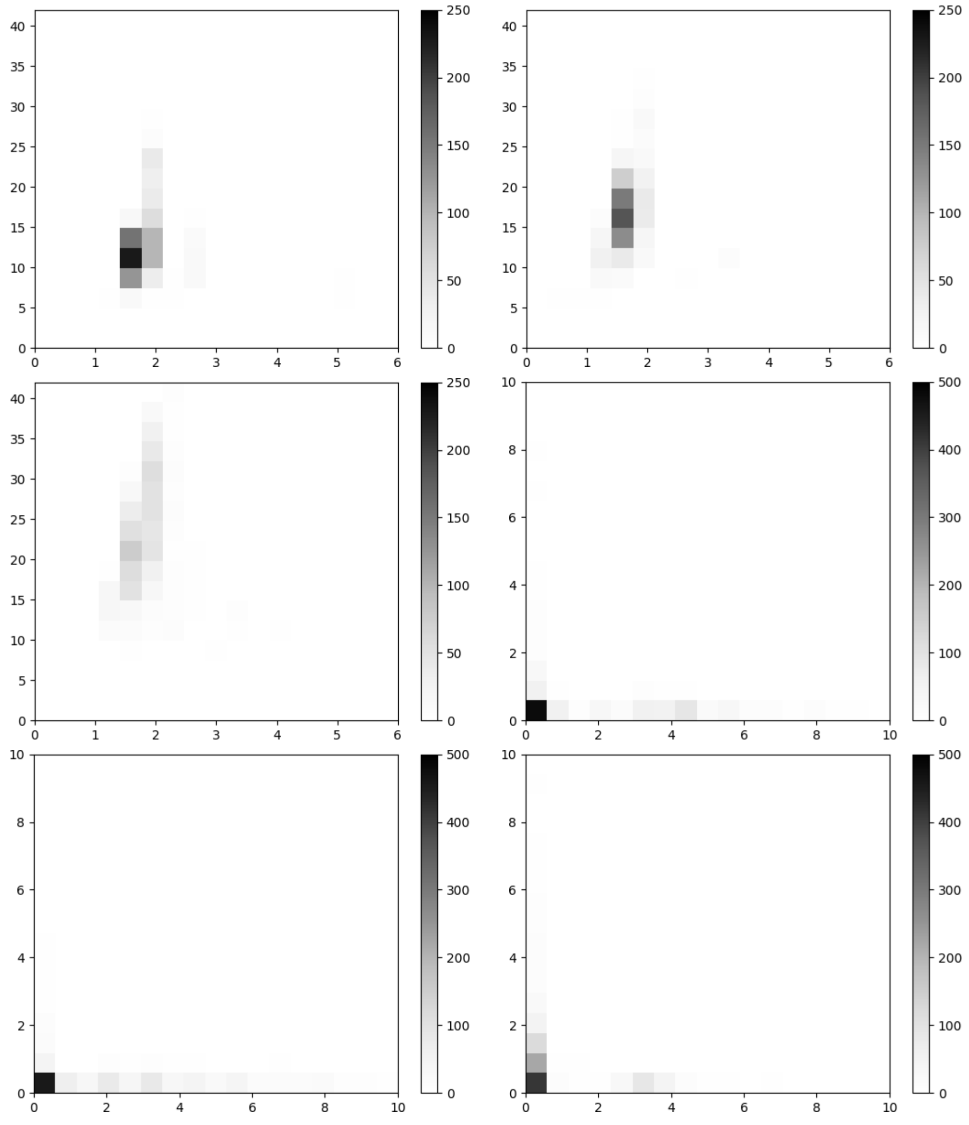


Figure 11: Heatmap histograms showing ‘fingerprints’ for each activity (from top-to-bottom, left-to-right: walking, walking upstairs, walking downstairs, sitting, standing and laying)

There is a significant increase in classification performance between the kNN algorithm trained on all-axes accelerometer data and the kNN algorithm trained on the single x-axis accelerometer data. This makes sense as there is three times as much data for the algorithm to make inferences from and the other two axes contain information the x-axis does not.

Looking at the confusion matrix for this classification, the predictions are crowded in the upper-left and bottom-right corner. This indicates that the algorithm has an easy time classifying between the three walking activities (walking, walking upstairs and walking downstairs) and the three non-walking activities (sitting, standing and laying) but a much harder time classifying between activities within these groups (e.g. walking vs walking upstairs).

The best performing classifier in this experiment was the SVM classifier which marginally outperformed the kNN equivalent by about 1% average precision. Both algorithms had the most difficulty classifying the 'sitting' and 'walking' activities (48-55% precision) and the most ease with the walking, walking upstairs, walking downstairs and laying activities (70-92% precision).

5.1.2 Frequency-domain 'binned' training (experiment 2)

This experiment was an extension on the previous one and tested whether increasing the amount of information about the Fourier transformed signal available to the classifier we could increase the classification precision. This did happen to be the case: this experiment gave results closest to the UCI HAR journal and outperformed all other tests on this dataset.

However, the trade-off with this approach is the processing time required for training and classification. The best performing algorithm (the 128-bin kNN classifier) took over 25 seconds to train and classify. Also, given the fact a kNN classifier must calculate the distances between points every time a new piece of data is to be classified (see disadvantages of kNN classifiers in section 2.3.2) this time becomes more relevant. Figure 12 shows a graph for precision divided by the time taken for different number of bins.

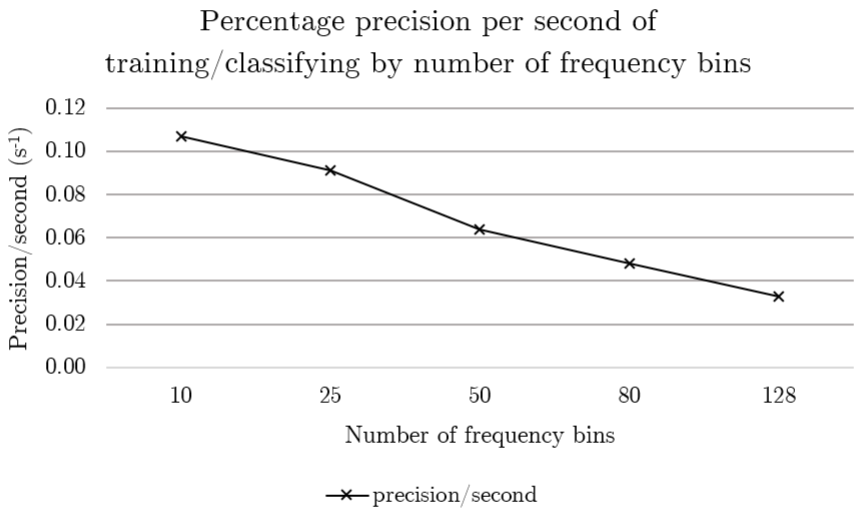


Figure 12: Graph showing precision per second by number of bins

This graph shows that while adding to the number of frequency bins continues to add to precision performance, the difference in precision per second of training/classifying gradually decreases. Therefore, if time of classification is a concern in a given application, it is worth reducing the number of bins as there is more value per second in fewer bins.

The SVM classifier for this experiment performed marginally worse than the kNN classifier but also took between 25-50% longer to train and classify. While it is not obvious why the SVM performed marginally worse it appears the precision classification for the 'sitting' activity dragged the overall average precision down as the precision for the other activities match those of the kNN classifier.

As mentioned in the analysis of the previous experiment, looking at the confusion matrices for this experiment tells the same story. The classifiers can easily tell between walking and non-walking activities but struggles more classifying within these groups.

5.1.3 Feature-based training (experiment 3)

This experiment aimed to replicate the approach used in the UCI HAR journal: extracting features from both the time-domain and frequency-domain representation of the signal and then training the algorithm from this feature vector. The classification matrix of an SVM classifier trained on the feature vector provided as part of the UCI HAR dataset can be seen in table 24.

Table 24: Classification report for SVM classifier trained on feature vectors provided by UCI HAR dataset

Activity	Precision	Recall	F1-score	Support
1. Walking	0.98	0.99	0.98	260
2. Walking upstairs	0.96	0.98	0.97	212
3. Walking downstairs	0.98	0.94	0.96	198
4. Sitting	0.93	0.84	0.88	274
5. Standing	0.85	0.93	0.89	249
6. Laying	0.99	1.00	0.99	278
Average/total	0.95	0.94	0.94	1471

A comparison of precision achieved by activity for an SVM classifier between experiment 3, the feature vectors provided in the UCI HAR dataset files and the UCI HAR journal can be seen in table 25.

Table 25: Comparison between precision achieved by experiment 3 and the UCI HAR journal

Activity	Experiment 3	UCI HAR feature vectors	UCI HAR journal
1. Walking	0.85	0.98	0.96
2. Walking upstairs	0.86	0.96	0.98
3. Walking downstairs	0.93	0.98	0.99
4. Sitting	0.43	0.93	0.97
5. Standing	0.59	0.85	0.90
6. Laying	0.66	0.99	1.00
Average/total	0.70	0.95	0.96

As can be seen in this comparison, the precision is worst when using the feature vector generated by experiment 3. Beyond this the classifier trained on the feature vectors provided within the UCI HAR dataset files is next best only slightly outperformed by the classifier used in the journal. Reasons why this is the case ultimately come down to two main factors: the feature vector and tuning of the SVM.

Regarding the feature vector used in training, the journal states that “[a] total of 561 features were extracted to describe each activity window”. This compares to the mere 48-value feature vector used in experiment 3. While the aim was to replicate the approach taken by the journal it was impossible to know exactly what features were being extracted from the sensor data and the table in the journal didn’t provide enough information to create a 561-value feature vector from its description. The feature vector used in the ‘UCI HAR feature vectors’ experiment was identical to that used in the journal.

Regarding tuning, the journal states “the SVM hyperparameters are selected through 10-fold Cross Validation procedure and Gaussian kernels are used for our experiments”. No tuning was done on my SVM classifiers, the default parameters for the `scikit-learn svm` were used. This lack of tuning explains the marginally lower precision in the ‘UCI HAR feature vectors’ experiment compared to the UCI HAR journal results. While tuning may have slightly improved the precision in experiment 3 the main improvement would definitely have been a more extensive feature vector as discussed prior.

5.1.4 Addition of gyroscope data (experiment 4)

The gyroscope-only experiments have a slightly lower classification precision compared to the accelerometer-only experiments for both classifiers, the only exception being experiment 2 with a kNN classifier where the gyroscope-only signal slightly outperformed the accelerometer-only signal.

Furthermore, the combination of the gyroscope and accelerometer signals improved on the accelerometer/gyroscope-only experiments in every case, the only exception being that the accelerometer-only signal slightly outperformed the combined signal in experiment 3.

However, while the combination has the best classification precision it is important to take note of the amount of extra data being used for training and classifying with only a marginal improvement in precision. In applications where speed of classification is important the accelerometer-only route may be a better choice.

5.2 MobiFall and MobiAct

5.2.1 Fall Detection

The fall detection proved to be a much easier task than the activity classification which was as expected. The first experiment was to feed the SVM classifier the full time-domain signal data for each activity window with labels as to whether the window did or did not contain a fall. The classifier returned perfect precision with this simple test, so the next challenge became optimising this algorithm to improve its speed and efficiency.

The SVM classifier generally seemed to be the slowest of the three with the kNN and DT tree classifiers being equally fast. A process of gradually removing features provided the discovery that the standard deviation of the signal alone (one value per activity window) was enough to classify a fall or non-fall. Figure 13 shows visually how this is the case. There is a near perfect split at a standard deviation of around 1 where non-fall (grey) windows have a lower standard deviation and falls (red) have higher.

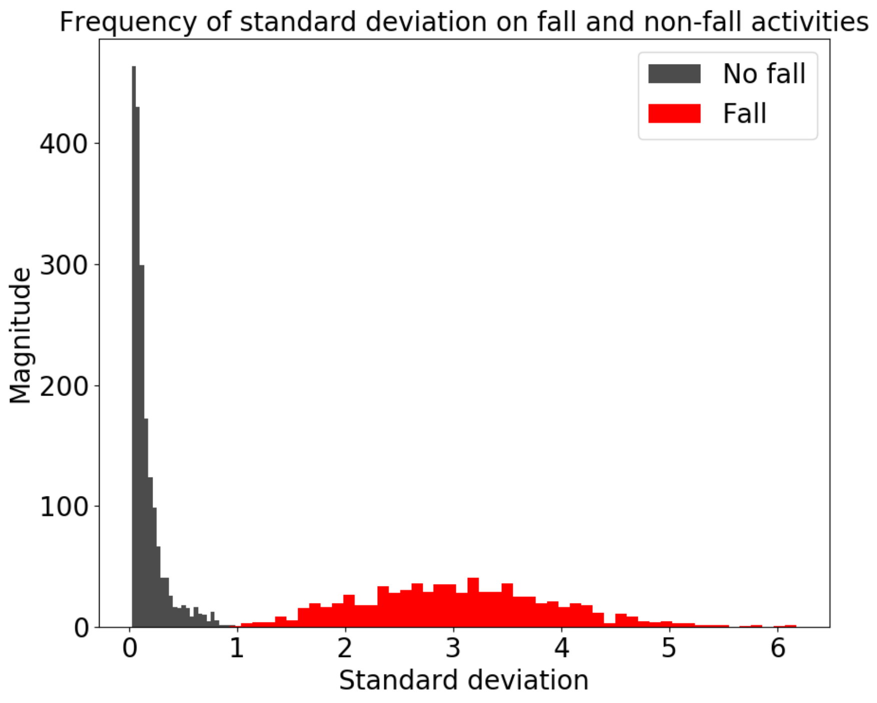


Figure 13: Histogram to show separation of standard deviation for fall and non-fall activities

5.2.2 Activity Classification

The SVM classifier could not be used for this part of the experiment as it would return an warning and not predict certain classes when tasked with certain activities. The warning message can be seen in listing 7.

Listing 7: SVM classifier warning

```
UndefinedMetricWarning: Precision and F-score are ill-defined and
    being set to 0.0 in labels with no predicted samples.
```

This warning is occurring due to the large imbalance of classes: there are significantly fewer walking upstairs (≈80), walking downstairs (≈80) and sitting (≈25) samples than walking (≈350) and standing (≈350) so the classifier is not predicting these labels whatsoever.

Table 26 shows a comparison between precision classification achieved on the two datasets for each experiment. The classifier used in this comparison is a kNN throughout.

Table 26: Comparison of classification precision between MobilFall and UCI HAR dataset classification

Experiment	MobiFall	UCI HAR
1	0.79	0.69
2	0.99	0.83
3	0.74	0.68

As can be seen in this table, the results achieved from the MobiFall dataset are considerably better than those achieved by the UCI HAR dataset. While it is not clear exactly why this is, two main factors play a role: window length and number of activities.

The first and perhaps the biggest reason results from the MobiFall dataset are better than those from the UCI HAR dataset is to do with the length of activity windows. Activity windows in the UCI HAR dataset were all 2.56-seconds in length whereas the activity windows used in the MobiFall classifier were all 10-seconds in length. This means the classifier had four times as much data to train and classify on for the MobiFall dataset which will obviously result in better classification and precision.

Another reason is to do with the number of activities. The MobiFall activity detection was only classifying between 5 different activities opposed to the 6 activities in the UCI HAR experiments. This reduces the complexity of the problem by a significant amount as once the algorithm has decided the window is non-walking there are only two choices left, sitting or standing rather than the three for the UCI HAR classifier.

All in all, the best performing classifier in this section (all bins, kNN classifier) performed extremely well with over 90% classification precision for all activities and only miss-classified 10 of the 895 test windows passed through it.

6 Conclusion

In conclusion, human activity recognition (HAR) is a very valuable tool with a huge range of uses and applications. This project has highlighted some of the struggles with producing machine learning algorithms to classify human activities but has also been highly successful in creating several examples of high-performing classifiers: not only in activity classification but fall detection as well. The project has been expansive: using three different machine learning algorithms, three different approaches to the problem and tested on two different datasets.

If it were possible to do further research, it would be useful to explore the following:

- Implementing the algorithm on-board a mobile device for classification at the point of data collection

- Testing different machine learning algorithms: for example, neural networks
- Testing the algorithms on further datasets to determine robustness and generality

References

[1] “Falls: applying all our health.” https://www.gov.uk/government/publications/falls-applying-all-our-health/falls-applying-all-our-health, Jun 2018. Accessed: 2019-03-09.

[2] J. Gallagher, “Inactivity ’kills more than obesity’.” https://www.bbc.co.uk/news/health-30812439, Jan 2015. Accessed: 2019-03-09.

[3] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, *A Public Domain Dataset for Human Activity Recognition Using Smartphones.* 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2013.

[4] G. Vavoulas, M. Pediaditis, C. Chatzaki, E. Spanakis, and M. Tsiknakis, “The mobifall dataset:: Fall detection and classification with a smartphone,” *International Journal of Monitoring and Surveillance Technologies Research*, vol. 2, pp. 44–56, 07 2016.

[5] O. D. Lara and M. Labrador, “A survey on human activity recognition using wearable sensors,” *Communications Surveys and Tutorials, IEEE*, vol. 15, pp. 1192–1209, 01 2013.

[6] A. Wang, G. Chen, J. Yang, S. Zhao, and C.-Y. Chang, “A comparative study on human activity recognition using inertial sensors in a smartphone,” *IEEE Sensors Journal*, vol. 16, pp. 4566–4578, 2016.

[7] K. Davis-Owusu, E. Owusu, V. Bastani, L. Marcenaro, J. Hu, C. Regazzoni, and L. Feijs, “Activity recognition based on inertial sensors for ambient assisted living,” 07 2016.

[8] A. Pachi and T. Ji, “Frequency and velocity of people walking,” vol. 83, pp. 36–40, 02 2005.

[9] The SciPy community, “Scipy v1.2.1 reference guide: scipy.fftpack.fft.” https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.fft.html\#scipy.fftpack.fft. Accessed: 2019-02-18.

[10] R2D3.us, “A visual introduction to machine learning.” http://www.r2d3.us/visual-intro-to-machine-learning-part-1/. Accessed: 2019-02-20.

[11] Mayur Kulkarni, “Decision trees for classification: A machine learning algorithm.” https://www.xoriant.com/blog/wp-content/uploads/2017/08/Decision-Trees-modified-1.png, 2017. Accessed: 2019-02-20.

[12] Antti Ajanki, “kNN Classification Diagram.” https://commons.wikimedia.org/wiki/File:KnnClassification.svg, 2007. Accessed: 2019-02-24.

[13] Y. Zhao, M. R. Kosorok, and D. Zeng, “SVM diagram from ’Reinforcement learning design for cancer clinical trials’,” 2009.

[14] “Classification report.” https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html. Accessed: 2019-03-12.