Classification of Human Motion Time Series Data

Luke Davidson - davidson.lu@northeastern.edu

CS 6140 - Machine Learning

October 30, 2022

Objectives and Significance

The use, evolution and importance of using time series data for classification methods has grown tremendously in recent years. Time series data classification methods are applied in industries ranging from healthcare, to fitness, to financial institutions and are used for applications such as the analyzation and classification of heart defects, mapping IMU data from fitness watches to specific types of workouts, and monitoring and classifying trends of prices of securities in the financial industry. The results of these applications lead to numerous benefits of varying effects, whether it be the ability to take optimal actions in the stock market given a certain observation and state, or being able to better understand your bodily habits and needs, or, in some cases, being able to detect a life saving diagnosis. The ability to accurately classify time series data leads to powerful analyses by simply measuring, monitoring, and drawing conclusions from normal everyday motion.

In this study, I propose to study the classification of human motion time series data. More specifically, my overall objective is to be able to correctly detect and classify the exact type of motion an individual is performing based on three-dimensional IMU position data from the head, arms, and legs of an individual performing a known action. As stated above, the classification of human motion can lead to numerous benefits within the health and fitness industries. Athletes of all levels, especially professional athletes, push their bodies to the limit by going through rigorous training sessions in order to maximize their performance for their given sport. Oftentimes these training sessions occur every day, or even multiple times a day. Understanding quantities such as performance measurements of an individual workout, or the level of strain that an athlete, or any individual, is putting on their body, is crucial to reaching their maximum potential and overall goals. In the health and fitness industry, much of the data that is used to quantify these values is obtained in the form of time series data, whether it be an IMU device, heart rate monitor, or a collection of multiple sources of data. As a former college athlete, I have always been interested in how I can apply my passion for both data manipulation and machine learning to my passion for fitness and athletics. Throughout this study, I aim to

relate those two passions to draw conclusions on the capability and limitations of classifying human motion time series data through various machine learning techniques.

Background

Time series classification (TSC) methods have been an area of increasing popularity amongst researchers and engineers over the last decade or so, mainly due to their wide variety of uses across many industries. TSC problems are typically solved by supervised learning approaches. The overarching goal when facing the problem of classifying human motion is to learn the different patterns of movement amongst each labeled class through labeled data, then detect those learned patterns and features in new input data to correctly classify that new data as a specific type of motion. Although the overall goal is the same, multiple different algorithms and methods of classification have proven to be effective when used to solve human motion TSC problems, and TSC problems in general, with some of these methods proving more effective than others depending on the problem at hand and the input data being used. The two main branches of algorithms used in TSC problems are those of instance based approaches and model based approaches [1]. Instance based approaches directly compare new data points to the training data and classify it based on similarities of some sort of quantification. When working with time series data, some instance based classification approaches include k-nearest neighbors (k-NN) and shapelet transformations. Model based approaches aim to compare new data points to some sort of decision rule created from training data to be able to accurately classify it. Popular model based algorithms used in TSC problems include support vector machines (SVM), training neural networks, and decision tree induction.

The k-nearest neighbor (k-NN) classification algorithm is often thought of as the simplest and most effective classification algorithm used to solve TSC problems. The main assumption of the k-NN algorithm is that data points that are similar in a measurable similarity metric belong to the same class. A similarity metric is calculated for each new input data point, and that metric is compared to the labeled training data. The *k* nearest training data points closest to that of the unlabeled input data metric are then selected, each having a known class label, and a class is selected based off of those labeled classes. A very important feature of implementing the k-NN algorithm with time series data is choosing the similarity metric that will be compared. Numerous types of similarity measurements have been used in studies throughout the recent years. Some of these measurements include shape search approaches, where different shapes and patterns in the time series data is mapped to quantities, discrete Fourier transforms, where data is

transformed in to the frequency domain and comparisons are made between most commonly occurring frequencies in the data, and subsequence mapping approaches, where time distances between recognizable subsequences of the original time series sequence are compared for similarity [1]. The k-NN algorithm has proven to be very effective in classifying time series data, although limitations such as computation cost and time come along with it.

Another instance based algorithm that has been studied more recently is that of shapelet transformations. Shapelets of a time series sequence are defined as subsequences that are in some sense maximally representative of a class. This relates to the above subsequence mapping approach. Shapelets for a respective class are typically learned from a gradient descent learning algorithm based on labeled training data. These class representative shapelets can then be compared to new unlabeled sequences to assess whether the given shapelet is present in the respective test sequence. If it is, it is assumed to belong to the class of the training shapelet. These comparisons are done by simply minimizing the sum of squared distance between the shapelet and a time period of the test sequence, assuming proper data pre-processing steps to normalize each subsequence. Shapelet transformations aim to assess the limitations that other popular TSC methods face, such as computational cost and time, while allowing for an increase in robustness due to the use of local features over global features.

Model based approaches, such as support vector machines (SVM), neural networks and decision tree induction, have also been used in TSC applications. SVM approaches work similarly to k-NN approaches, although instead of directly comparing new data points to the training data, a decision boundary is created during the training phase and new data points are compared to that decision boundary. Similarity measurements for time series sequences are calculated using similar methods as described above in the k-NN approach, using methods such as shape searches, Fourier transforms, and subsequence mapping [3]. When using SVM approaches, class labels are binary. That is, new input data will either belong to one class or the other. A frequent approach to solve this limitation in machine learning is the use of deep neural networks (DNNs). DNNs consist of multiple layers where many different weights and parameters are calculated and tuned based on labeled training data, where weights signify the likely presence of a respective feature. The output of the last of these layers is the likelihood that the input data point belongs to each respective class label. DNNs have been widely used for applications such as image and language classification, but have become increasingly popular in TSC applications in recent years due to promising studies. Compared to other TSC methods, DNNs provide accurate results in a fraction of the time.

Proposed Approach

In this study, I propose to implement classification methods able to classify specific human motions such as punching, walking, and waving. I have obtained a data set from the UCI Machine Learning Repository that contains sensor data from 10 subjects performing 20 different actions. Each subject was equipped with nine IMU sensors capturing their motion: one on the head, and two on each arm and leg. The subjects were guided through performing specific actions while data was collected. The action set includes 10 “aggressive” actions, and 10 “normal actions”. The aggressive actions are elbowing, front kicking, hammering, headering, kneeing, pulling, pushing, punching, side kicking and slapping, while the normal actions are bowing, clapping, handshaking, hugging, jumping, running, eating, standing, walking and waving. Each subject performed each action for approximately 10 seconds, with data being sampled at approximately 200 Hz, leading to roughly 2000 data points for each action.

My goal is to utilize the above classification algorithms, specifically k-NN and shapelet transformations, to be able to correctly classify actions based on positional sequence data inputs. The general process for implementing the k-NN and shapelet algorithms using time series data includes: 1. Identifying patterns of subsequences for each action using labeled training data, 2. Comparing these identifying subsequences to those of new unlabeled sequences by calculating an error function relating the two subsequences to determine the likelihood of a relationship, 3. Identifying the top *k* matches of the input data, and 4. Making a decision of the class of the unknown data point being analyzed based on the known classes of the top *k* matches.

Before I complete the steps listed above, the first thing I’ll do in my implementation will be preprocessing the data to get it ready for the algorithm used. Potential steps in the preprocessing stage of my implementation include normalization, smoothing, shifting and resizing of the sequences. The comparison calculation I plan on implementation will be a minimization of error between the known training subsequence and the unknown test subsequence. The normalization step in the preprocessing phase is extremely important because it is important for the sequences to be scale invariant. For example, the time series sequences of positional leg data for a taller individual versus a shorter individual kicking an object will likely have similar shape but different scales. Both of the sequences will be representative of the action “kicking”, so we want the error calculator to be low. The normalization of the data in the preprocessing phase aims to eliminate the limitation of scaling. Normalization also helps eliminate mathematical errors of exploding or vanishing gradients in

the gradient descent search to locate shapelets, further described in the feature extraction phase below [4]. The shifting and resizing of data sequences will mostly be addressed in the feature extraction and comparison phases of my implementation.

In order to compare new unlabeled sequences to those of known sequences, we will want to extract features in both that are highly representative of each individual action, or label. This part of my implementation is most representative of the shapelet transformation algorithm. There are multiple methods and algorithms for extracting shapelets from test data. Some of these include the brute force search, gradient descent methods and frequency analysis. The brute force search involves creating a list of potential shapelet candidates by slicing up whole sequences into subsequences of a certain size. This size and number of subsequences selected are represented by the hyperparameters *window size* and *step size*, respectively. Each candidate for a subsequence is then compared to the known shapelet to determine the most likely location of the shapelet in the subsequence, and how likely it fits. Based on the location, a further method of correct shifting will be applied to ensure the shapelet windows are best shifted, further explained below in the comparison phase. There are also prebuilt shapelet identifier methods in python libraries, specifically the python library tslearn. This method uses a gradient descent approach to locate the window corresponding to the strongest weights, or matches, of an inputted shapelet. I plan to spend a significant amount of time in my implementation on this phase, working on and comparing multiple methods of shapelet identification methods, as the selection of identifiable features is the backbone to obtaining optimal classification results.

The comparison phase of my implementation involves the calculation of error between two representative subsequence shapelets. A simple example of an error calculation is that of minimizing the sum of squared error between two points. Figure 1 shows an example of this error calculation:

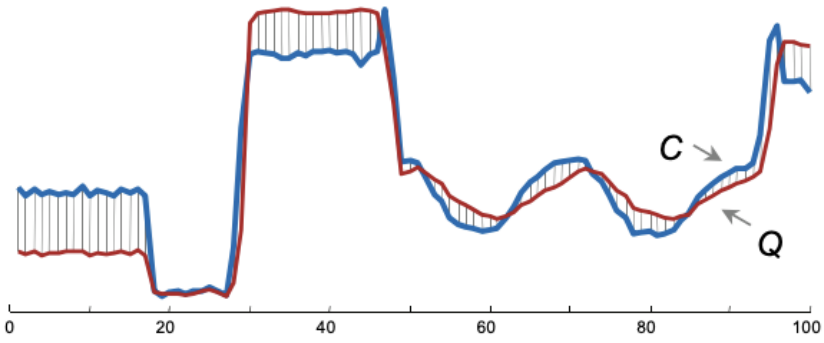

Figure 1: Two subsequence shapelets, C and Q, representing the same class

In figure 1, let us assume that subsequence C is the learned shapelet from the known labeled training sequence, and Q is the extracted shapelet from an unlabeled test sequence of the same action. The error between the two sequences will be the sum of squared distances between two, represented as the gray vertical lines in figure 1. After data preprocessing, we can see that two shapelets of similar classes will have a much smaller sum of squared distances than two that are completely different. There are a couple things to keep in mind when calculating an error function such as the sum of squared distances. The first is that shapelets extracted will need to be the same size. This parameter is often known as the *window size* when working with time series data [4]. This can be thought of as a small preprocessing step where the data is resized to fit the given algorithm. Another thing to keep in mind is that the shapelets will need to be shifted properly in order to obtain the lowest possible error. If sequence C was shifted to the left, the error calculation would become much larger and thus cause bias in the final decision. Proper shifting of shapelet subsequences largely confronted in the feature extraction phase of my implementation.

The final phases of my implementation will involve the decision and prediction phases and will be most representative of the k-NN algorithm. Once the test sequence has been compared to the training points via an error calculation, our main goal is to minimize this error metric. The k-NN algorithm does this by selecting the lowest *k* errors as fitting sequences, where *k* is an integer. The reason for multiple selections of potential classes is to make the algorithm more robust to outliers. The prediction phase of the algorithm makes a final prediction on the class of the unlabeled sequence based on the classes of the *k* sequences with the lowest error found in the comparison phase. There are multiple types of prediction algorithms for the k-NN algorithm, the simplest being an average over the *k* sequence matches. Another frequently used prediction algorithm is weighted predictions, where subsequences of lower error carry more weight into the final decision prediction [5]. I aim to test multiple of these decision algorithms to determine the variability and robustness of correctly classifying the sequences.

The post implementation phase of my implementation will simply include the percentage of correct classifications. A unique aspect of my study will be studying the motion of each limb separately to see the accuracy potential of each classification algorithm with respect to each limb. For example, one might imagine the difference in the motion of an arm during the actions elbowing and pushing will be much greater than that of the difference in movement of the legs. My initial hypothesis is that some action classifications will be much more accurate for certain limbs, such as pushing with the arms and kicking with the legs.

If I face errors in my initial plan of an implementation, some alternate directions I can go in are generalizing my problem statement to simply classifying aggressive behavior versus normal behavior, or limiting the sequences I use based on general knowledge of the action, for example, only using upper body sensor data for actions mainly involving the upper body. Generalizing my problem statement to classifying aggressive behaviors versus normal behaviors simplifies my implementation because it limits it to a binary classification problem. It is under the assumption that all aggressive behaviors will show patterns of similar shape, maybe a sudden quick movement, while normal behaviors will show patterns similar to each other, maybe a slower, more smooth movement. Thus, error calculations in relation to each other will likely be much lower for the given correct class, than the second incorrect class. I also anticipate that I may face errors in classifications due to the many degrees of freedom in my initial dataset. As mentioned above, this relates to the thought that there will not be many distinct features in the data from one of the leg sensors for the actions that mainly involve the upper body. This will lead to a greater possibility of incorrect labels having low errors, because no strong distinctive shapelet features were able to be detected from the training data. Limiting the data I use to only using the sequences of data from limbs corresponding to the action will likely increase my evaluation and prediction percentage because those sequences are most likely to have the strongest identifying features. Sequences of arm movement during a punch will likely be of high correlation to one another, while leg movement of a punch has a lot more room for error.

Individual Tasks

Since I am working on this project alone, this section boils down to simply everything! As mentioned in the opening section, I have always been extremely interested in relating my two passions of data manipulation and machine learning to my passion for fitness and movement, and I am very excited to work on this project statement. I will first spend time working on the data preprocessing phase, making sure it is stored and in the correct format that will maximize my implementation efficiency. I then plan on spending the majority of time on feature extraction, recognizing patterns in the sequences that are most representative of the given class. Finally, I will implement multiple methods of decision making and tune parameters such as window size and k to determine parameters that minimize my classification error.

References

[1] Theodoridis, Theo. School of Computer Science and Electronic Engineering; University of Essex. https://archive.ics.uci.edu/ml/datasets/Vicon+Physical+Action+Data+Set

[2] Yen-Hsien, Lee. Chih-Ping, Wei. Tsang-Hsiang, Cheng. Ching-Ting, Yang. 10 January 2010. National Taiwan University, Taipei. https://www.sciencedirect.com/science/article/pii/S0167923612000097

[3] Timothy R. Dinger, Yuan-chi Chang, Raju Pavuluri, Shankar Subramanian. January 26, 2022. Time Series Classification. https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification/

[4] L. Ye & E. Keogh. Time series shapelets: a new primitive for data mining. SIGKDD 2009. https://tslearn.readthedocs.io/en/stable/user_guide/shapelets.html

[5] Lexiang Ye. Eamonn Keogh. University of California, Riverside. Time Series Shapelets: A New Primitive for Data Mining. https://www.cs.ucr.edu/~eamonn/shaplet.pdf

[6] Rutuja Pawar. 15 June 2021. k-NN based Time Series Classification. https://towardsdatascience.com/k-nn-based-time-series-classification-e5d761d01ea2

[7] Gustavo E.A.P.A. Batista. Xiaoyue Wang. Eamonn J. Keogh. University of California, Riverside. A Complexity-Invariant Distance Measure for Time Series. https://www.cs.ucr.edu/~eamonn/Complexity-Invariant%20Distance%20Measure.pdf