MOVELETS: Exploring Relevant Subtrajectories for Robust Trajectory Classification

Carlos Andres Ferrero
Universidade Federal de Santa Catarina (PPGCC/UFSC)
and Instituto Federal de Santa Catarina (IFSC)
Florianopolis and Lages, SC, Brazil
andres.ferrero@ifsc.edu.br

Luis Otavio Alvares
Universidade Federal de Santa Catarina (UFSC)
Florianopolis, SC, Brazil
luis.alvares@ufsc.br

Willian Zalewski
Universidade Federal da Integração
Latino-Americana (UNILA)
Foz do Iguaçu, PR, Brazil
willian.zalewski@unila.edu.br

Vania Bogorny
Universidade Federal de Santa Catarina (UFSC)
Florianopolis, SC, Brazil
vania.bogorny@ufsc.br

ABSTRACT

Several methods for trajectory classification build models exploring trajectory global features, such as the average and the standard deviation of speed and acceleration, but for some applications these features may not be the best to determine the class. Other works explore local features, applying trajectory partition and discretization, that lose important movement information that could discriminate the class. In this work we propose a new method, called Movelets, to discover relevant subtrajectories without the need of a predefined criteria for either trajectory partition or discretization. We extend the concept of time series *shapelets* for trajectories, and to the best of our knowledge, this work is the first to use shapelets in the trajectory domain. We evaluated the proposed approach with several categories of datasets, including hurricanes, vehicles, animals, and transportation means, and show with extensive experiments that our method largely outperformed state of the art works, indicating that Movelets is very promising for trajectory classification.

CCS CONCEPTS

•Information systems → Geographic information systems; Data mining;

KEYWORDS

Spatio-temporal data analysis, trajectory classification, trajectory shapelets, relevant subtrajectories, movelets.

ACM Reference format:
Carlos Andres Ferrero, Luis Otavio Alvares, Willian Zalewski, and Vania Bogorny. 2018. MOVELETS: Exploring Relevant Subtrajectories for Robust Trajectory Classification. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 8 pages.
DOI: 10.1145/3167132.3167225

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC 2018, Pau, France
© 2018 ACM. 978-1-4503-5191-1/18/04…$15.00
DOI: 10.1145/3167132.3167225

1 INTRODUCTION

Trajectory classification is an important data mining task, since it is used to discover the category of the trajectories or the moving object, as for instance the transportation means of an individual such as car, bike, or pedestrian [2, 10, 13], the profile of a person such as retired, worker, student, etc. [1], if an object is stopped or moving [7], the strength level of a hurricane [5], the type of an animal [5, 8], and so on. Trajectory classification is not a trivial task, since the input data is in general a sequence of spatio-temporal points. So the first challenge in trajectory classification is to define a set of relevant features to discriminate the class, that may be related to a single trajectory point, a trajectory part, called *subtrajectory*, or to the entire trajectory. The second challenge is to select the most relevant features to build good classification models.

Some methods for trajectory classification as [10, 13] build models exploring trajectory global features, such as average speed, average acceleration, standard deviation, etc. However, these features may not be the best to determine the class of a trajectory for some applications. Other works as [2, 5, 8] explore local features, but applying trajectory partition and discretization techniques that lose important movement information that may be determinant for the class.

We claim that *relevant subtrajectories*, which are more discriminative for classification problems may not be discovered by methods that predefine the criteria for trajectory partition or discretization, as has been done in the works of [2, 5, 8, 10, 13]. A relevant subtrajectory is a part of a trajectory with the capability for discriminating the classes of the classification problem. Figure 1 shows an example of a relevant subtrajectory (blue line) for classifying hurricane trajectories (classes green and red). Note that trajectories passing near the blue line are all of the same class (green).

A novel concept applied with success on time series classification that allows the identification of relevant parts of time series without performing either partition or discretization is the *Shapelet Analysis*. This concept was introduced in [11] and explored and improved in [4, 6, 12]. However, *Shapelet Analysis* cannot be directly applied to trajectory data, because it is defined for only one variable over time, and trajectory data have two variables to represent the space over time: x and y.

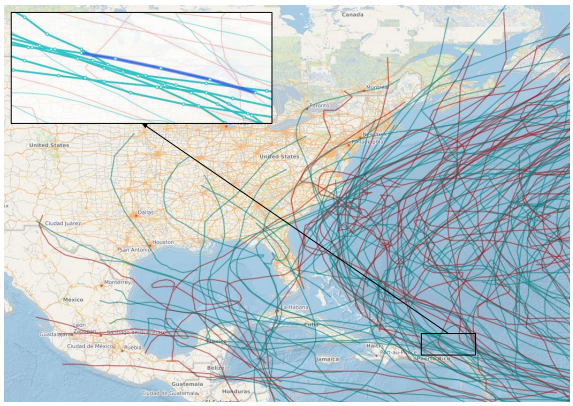

Figure 1: An example of relevant subtrajectory.

In this paper we propose a new parameter free method based on *shapelet analysis* for trajectory data, called MOVELETS, in order to extract relevant subtrajectories to generate local features for robust trajectory classification. In summary, the main contributions of this paper include:

- A new method to discover relevant subtrajectories for trajectory classification, without a pre-defined criteria for trajectory partition, using the concept of *Shapelets*. To the best of our knowledge this is the first work using *shapelets* in trajectory data.
- A new pruning method to reduce the relevant subtrajectories to a number lower than the total number of trajectories.
- An extensive experimental evaluation and comparison to state-of-the-art over six datasets to demonstrate the potential of our method.

The remaining of this paper is structured as follows: Section 2 presents related work for trajectory classification, Section 3 describes the proposed method, Section 4 shows the experiments, and Section 5 conclusions and future work.

2 RELATED WORK

Existing works on trajectory classification consist of extracting global and/or local features of trajectories to build the classifier. On the one hand, *global features* consider the entire trajectory to capture movement information. Several works use descriptive statistics (e.g. mean, standard deviation, skewness) of *point features*[1], such as acceleration, turning angle, and straightness [2, 10]. Zheng [13] proposes the use of three global features for classifying transportation modes: heading change rate, stop rate, and velocity change rate. Global features have shown good results for classifying transportation modes, but for general purposes the main limitation is that an object does not perform a uniform movement in the entire trajectory.

On the other hand, *local features* consider trajectory parts, called *subtrajectories*, to offer more detailed information about movement. Dodge [2] proposes a method to extract local features by transforming the trajectory into times series of *point features*. The time series are discretized into low and high deviation and sinuosity values, forming the trajectory Profile Decomposition (PD), and mean, standard deviation, and proportion from PD discrete values are computed as local features. This approach is limited because *discretization* can hide important movement information, as for instance, when a part of a trajectory is characterized by low speed deviation, the information about how much low speed deviation is, is lost.

Lee in [5] proposes a method that initially partitions trajectories into subtrajectories when the movement direction changes rapidly. In a second step, this work groups subtrajectories with similar movement, and each group may have subtrajectories that belong to different class labels. Then, the subtrajectories are split in order to create subtrajectory groups of only one class label. After that, the groups are evaluated according to the group relevance, which is measured by the average distance from a group to other groups of different classes, where the larger the distance the higher is the relevance. Thus, only those groups with a relevance value greater than the median relevance are selected. Finally, the trajectories are represented by a feature vector, where each value corresponds to the frequency of how many times each relevant subtrajectory occurs into a trajectory. While Lee [5] used only the *space dimension*, Patel [8] considered also the *time*. But, these two methods are limited at the first step, when the trajectory is partitioned into subtrajectories by direction change [5] or traveled distance [8]. At this partition, it is assumed that any local feature extracted over the subtrajectory, as for instance the average speed or traveled distance is uniform. However, if the subtrajectory has low speed at the beginning and high speed at the end, this information is lost.

3 MOVELETS: FINDING RELEVANT MOVEMENT SUBTRAJECTORIES

In this section we detail our proposal for finding discriminative and relevant parts of trajectories for trajectory classification without a predefined criteria for trajectory partition or discretization. We extend the concept of *time series shapelets* to trajectory *movelets*.

3.1 Basic Definitions

A *trajectory* $T = \langle p_1, ..., p_m \rangle$ is a sequence of time-ordered points $p_i = (x, y, t)$, where x, y correspond to the spatial location of the object at the time instant t.

Our goal is to find parts of trajectories with high discriminative power among classes. A part of a trajectory is called *subtrajectory*. So given a trajectory T of length m, a *subtrajectory* $s = \langle p_a, \ldots, p_b \rangle$ is a contiguous subsequence of T starting at point p_a and ending at point p_b, where $1 \leq a < m$ and $a < b \leq m$. The length of the subtrajectory is defined as $w = |s|$. In addition, we also define the set of all *subtrajectories* of length w in T as S_T^w, and the set of all *subtrajectories* of all lengths in T as S_T^*.

One of the key aspects to find discriminative or relevant parts of a trajectory is the distance between two subtrajectories. To compute this distance, we need first to define how to measure the distance between two trajectory points. This distance may consider one or more dimensions, since a point may have several information, such as spatial location, time, speed, acceleration, direction, etc. For that reason, we formally define the concept of *distance between points*.

[1]Point feature is any information related to the points of a trajectory.

Definition 3.1. **Distance between two trajectory points**. Given two trajectory points p_i and p_j represented by d dimensions, the distance between these trajectory points $Dist_p(p_i, p_j)$ must consider their distance in each dimension, and combine them into a non-negative value that respects the property of symmetry, $Dist_p(p_i, p_j) = Dist_p(p_j, p_i)$.

The idea behind Definition 3.1 is to allow the use of a distance function for each point dimension (such as spatial location, time, speed, acceleration, direction, and so on) and combine them into a unique distance function to compute the distance between two subtrajectories of equal length. So the distance between two subtrajectories is given in Definition 3.2.

Definition 3.2. **Distance between two subtrajectories of equal length**. Given two subtrajectories s_1 and s_2 of equal length, $Dist_e(s_1, s_2)$ computes the distance between their sequential points, returns a non-negative value and respects the property of symmetry, $Dist_e(s_1, s_2) = Dist_e(s_2, s_1)$.

To find the most similar *subtrajectory* of a trajectory T to a *subtrajectory* s we calculate, for all subtrajectories of T of length $w = |s|$, the distance between each subtrajectory of T and s. So, the most similar subtrajectory of T to s (*best match*) is a subtrajectory r with the minimum distance. This comparison is given in Definition 3.3.

Definition 3.3. **Distance between trajectory and subtrajectory.** Given a trajectory T and a subtrajectory s of length $w = |s|$, this distance is the best match of s into T, which is defined by $SubDist(s, T) = min(Dist_e(s, r) \mid r \in S_T^w)$, where S_T^w is the set of all subtrajectories of length w into T, and $min()$ returns the minimum distance between s and all subtrajectories in S_T^w.

The number of possible subtrajectories of any length in a trajectory classification problem of n trajectories of length at most m is $O(n \times m^2)$. By representing trajectories using all subtrajectories as features, the induction of classification models is impracticable, because of the relation between instances and attributes. So, the selection of only the most *relevant subtrajectories* is necessary. We define a subtrajectory candidate in Definition 3.4.

Definition 3.4. **Subtrajectory Candidate**. A *subtrajectory candidate* is a tuple $(T, start, end, D, quality)$, where:

- T is the trajectory that origins the candidate;
- $start$ is the position in T where the candidate begins;
- end is the position in T where the candidate ends;
- D is a set of pairs (d, c), where d is the distance value to a trajectory T' (Definition 3.3) and c is the class label of T';
- $quality$ score is a relevance value.

To evaluate the relevance of each candidate we use the set of distances D and the concept of *orderline*. For example, let us consider the four trajectories shown in Figure 2, where T_1 and T_2 are of class A, and T_3 and T_4 are of class B. The distance of the candidate s to a subtrajectory of equal length in trajectory T_1 is zero, since this subtrajectory belongs to T_1. The shadow between the subtrajectory candidate and the trajectories in the figure shows the *distance between the best match of T_2, T_3, and T_4 to the candidate s (subtrajectory in blue).*

The distance between the candidate s and each subtrajectory is represented as an *orderline*. An *orderline* is a visual representation of the set of distances, where the values are ascendantly ordered. The distances d_1, d_2, d_3, d_4 in Figure 2 are displayed in an *orderline* from 0 (left) to $+\infty$ (right).

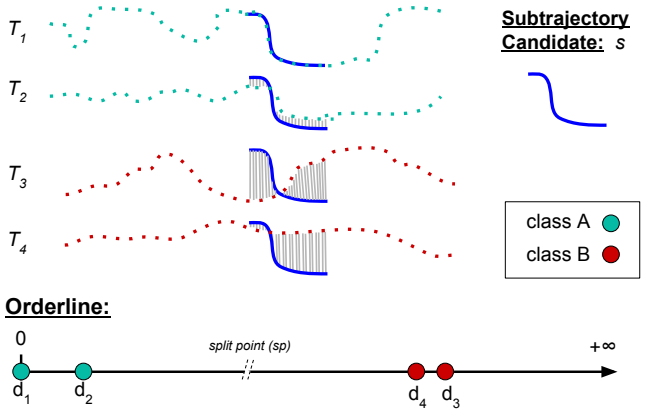

Figure 2: Example of a candidate *orderline*.

One way to measure the *quality* of a subtrajectory candidate is to define a *split point sp* to divide the *distance orderline* into two disjoint subsets of distances, *left* and *right*, to separate the trajectories that are closer and further to the candidate subtrajectory s. In Figure 2 the classes are well separated in the middle of distance values d_2 and d_4, where the subset *left* contains the distances of trajectories of class A and the subset *right* contains the distances of trajectories of class B. In real scenarios with many trajectories, in general, there is not a single subtrajectory and split point that solves the problem. So the subtrajectory quality is related to the capability to find a split point that better separates the classes. Based on the concept of *quality* we define a *movelet* as in definition 3.5.

Definition 3.5. **Movelet**. Given a trajectory T and a subtrajectory candidate $s \in T$, the subtrajectory s is a *movelet* if for each subtrajectory $r \in T$ that overlaps s in at least one point, $s.quality > r.quality$.

In other words, a candidate is a *movelet* if there is no other candidate overlapping it with more relevance.

Based on the previous definitions, in the following section we present the proposed method to discover movelets for trajectory classification.

3.2 Discovering Relevant Subtrajectories

We propose a three step method, called MOVELETS, for discovering relevant subtrajectories: (1) *movelet* discovery; (2) *movelet* pruning; and (3) *movelet* transformation. Hereafter we refer to MOVELETS as the method and *movelets* as the relevant subtrajectories.

Step 1: Movelet Discovery. The movelet discovery step consists of exploring all subtrajectory candidates from a trajectory training set and selecting only the most relevant. These relevant subtrajectories we call *movelets*. This step is detailed in Algorithm 1, that has as the unique input the trajectory training set **T**, without any parameter. The output is the set of movelets.

Algorithm 1 finds for each trajectory in the training set **T** the most relevant subtrajectories (lines 3-17). For each $T \in \mathbf{T}$ it computes all distances between all trajectory points of T and all points

Algorithm 1: MoveletDiscovery

```
   Input  : T // trajectory training set
   Output : movelets // set of relevant subtrajectories
1  movelets ← ∅;
2  n ← T.size;
3  for each trajectory T in T do
4      trajectoryCandidates ← ∅;
5      m_T ← T.length;
6      M_1 ← ComputePointDistances (T, T);
7      for each subtrajectory length w in (2, m_T) do
8          M_w ← CSD (M_{w-1}, M_1, n, m_T, w);
9          candidates ← GenerateCandidates (T, T, w, M_w);
10         SortByQuality (candidates);
11         RemoveSelfSimilar (candidates) // same length;
12         trajectoryCandidates ←
             trajectoryCandidates ∪ candidates;
13     end
14     SortByQuality (trajectoryCandidates);
15     RemoveSelfSimilar (trajectoryCandidates);
16     movelets ← movelets ∪ trajectoryCandidates;
17 end
18 return movelets
```

Algorithm 2: *GenerateCandidates*

```
   Input  : T // trajectory for generating candidates
            T // trajectory training set
            w // size of subtrajectory candidates
            M_w // distance matrix for size w
   Output : candidates // set of subtrajectory candidates
1  candidates ← ∅;
2  n ← T.size;
3  m_T ← T.length;
4  offset ← (w − 1);
5  for each position j in (1, m_T − offset) do
6      D ← ∅;
7      for each trajectory i in (1, n) do
8          minSum ← minValue(M[i, j, ..]) ;
9          distance ← √(minSum/w) ;
10         D[i] ← (distance, T[i].class) ;
11     end
12     quality ← AssessQuality (candidate.D);
13     candidate ← (T, j, j + offset, D, quality) ;
14     candidates ← candidates ∪ candidate ;
15 end
16 return candidates
```

of **T**, and stores them into the matrix M_1, in order to perform this computation only once (line 6). Next, it explores all subtrajectory lengths from 2 to m_T, one by one, by computing subtrajectory distances (internal loop in lines 7-13). For a length w it computes the distance matrix M_w, using the distance values computed for subtrajectories of sizes $(w-1)$ and 1, represented by M_{w-1} and M_1, respectively (line 8). Matrix M_w contains the sum of squares of point distances, where each *point distance* corresponds to the euclidean distance between spatial locations. Next, it generates the subtrajectory candidates based on M_w (line 9) (function *GenerateCandidates* is detailed later in Algorithm 2) computing their quality. After that, it sorts candidates by quality score (lines 10) and then it removes *self similar* candidates of size w to reduce the subtrajectory redundancy (line 11). Two candidates are *self similar* if they are overlapping on at least one point and the algorithm preserves the candidate with highest quality. Finalizing the internal loop, it adds the remaining candidates to *trajectoryCandidates* (line 12). Following the external loop, it sorts the trajectory candidates and removes those *self similar* (lines 14-15), and adds the remaining trajectory candidates to the *movelets* set.

Because of space limitations, we detail only the function *GenerateCandidates*, but the entire algorithm can be obtained from the authors website or by contact. The process of candidate generation is described in Algorithm 2.

Algorithm 2 obtains the subtrajectory candidates of length w of a trajectory T, as well as the best match distances of candidates on each trajectory. Considering each subtrajectory of length w in T starting at position j (external loop in lines 5-15) as a candidate, it defines the set of distances D (line 6) to store the distance of the best match of the candidate on each trajectory (lines 7-11). The distance is then normalized and stored into the set D (lines 9-10). After that, using D it computes the quality of the subtrajectory with the function *AssessQuality*(line 12) (function *AssessQuality* is detailed later in Algorithm 3) and instantiates the subtrajectory candidate (line 13).

A classical approach to measure the quality of a shapelet in time series consists of the maximum information gain [4, 11]. However, in trajectory data this approach returns high split point values in order to achieve the maximum information gain, increasing the movelet variability and decreasing the generalization. To avoid this problem, in this work we use the maximum information gain keeping the left side of the distance *orderline* pure, that we called *Left Side Pure* (LSP). The term *pure* indicates that all distances in the left side are of the same class. Algorithm 3 describes this process.

Algorithm 3: *AssessQuality LSP*

```
   Input  : D // set of distances
   Output : quality // quality value
1  orderline ← sort D by d's;
2  class ← orderline[1].class;
3  i = 2;
4  while orderline[i].class = class do
5      i ← i + 1;
6  end
7  sp ← (orderline[i].d + orderline[i − 1])/2;
8  quality ← InformationGain (orderline, sp);
9  return quality
```

Algorithm 3 starts sorting D in ascending order of distance values as the *orderline* (line 1). The 1st element on the *orderline* indicates the candidate class (line 2). So, starting with the 2nd element, the algorithm increases i while the ith element has the same class of the candidate (lines 3-6). After that, it calculates the split point and the respective information gain (lines 7-8).

Figure 3 shows a demonstration of *movelets* discovery for a Hurricanes dataset with two classes (green and red). Figure 3(a) shows

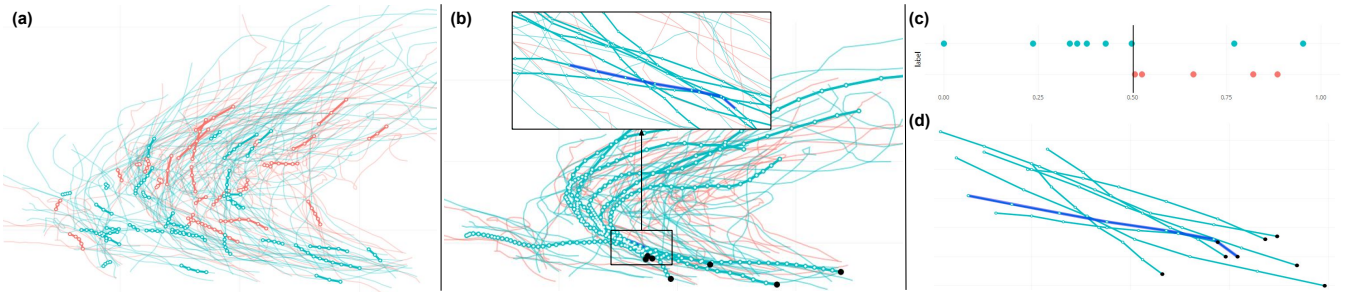

Figure 3: Demonstration of movelets discovery in a Hurricane dataset.

the *movelets* found for each class. Figure 3(b) shows an example of *movelet* of class green and focuses in the subtrajectory area, where the relevant subtrajectory is in blue and the green trajectories are those passing near the *movelet*. Figure 3(c) shows the distance *orderline* for the *movelet*, where the vertical line is the split point that separates the closer trajectories on the *left side* and the more distant trajectories on the *right side*. Note that by assessing the quality using LSP the algorithm keeps on the *left side* of the *ordeline* only distances to trajectories of the same class. Figure 3(d) shows the subtrajectories at the *left side* of the *orderline* in green and the *movelet* in blue.

Step 2: Movelet Pruning. The *movelets* found by Algorithm 1 are the relevant subtrajectories. However, the number of *movelets* found is $O(n \times m)$, but this is not good for inducing classification models, because of the relation between number the of instances (that are the trajectories) and attributes (*movelets*). To reduce the number of *movelets*, the pruning step removes redundant *movelets*. In this work, we consider a *movelet* as redundant if the trajectories on the left side of the *ordeline* were already covered by other *movelets* with better quality. Algorithm 4 describes this step.

Algorithm 4: *MoveletPruning*

Input : S // movelets ordered in descending order of quality
Output: S' // non-redundant movelets

```
1  S' ← ∅;
2  CoveredT ← ∅;
3  for each movelet s in S do
4      CoveredByMovelet ← Left side of s;
5      if (CoveredByMovelet ∩ CoveredT) ≠ ∅ then
6          S' ← S' ∪ s;
7          CoveredT ← CoveredT ∪ CoveredByMovelet;
8      end
9  end
10 return S'
```

Algorithm 4 starts initializing the set of non-redundant *movelets* S' and the set of covered trajectories *CoveredT* (lines 1 and 2). It analyzes the *movelet* one by one in descending order of quality (lines 3 to 9). It finds the trajectories on the left side of the distance *orderline* of *movelet* $S[i]$ and stores them into *CoveredByMovelet* (line 4). Only if the *movelet* S[i] covered at least one new trajectory it adds the *movelet* $S[i]$ to the set of filtered *movelets* S', and adds the new covered trajectories into *CoveredT* (lines 5 to 8). By considering only the non-redundant *movelets* we reduce their number to $O(n)$, because each *movelet* in S' covered at least one new trajectory not covered by other *movelets* with better quality in S'.

Step 3: Movelet Transformation. The *third step* of our method consists in representing a trajectory set using the *movelets* as attributes and the distance of each trajectory to each *movelet* as the attribute values. Algorithm 5 describes this step.

Algorithm 5: *MoveletTransformation*

Input : **T** // trajectory set
S' // *movelets*
Output: *dataset* // an attribute-value representation

```
1  dataset ← ∅ ;
2  for trajectory i in (1, T.length) do
3      for movelet j in (1, S'.length) do
4          distance ← SubDist(S'[j], T[i]);
5          maxDistance ← S'[j].quality.maxDistance;
6          sp ← S'[j].quality.sp;
7          if distance ≤ sp then
8              normDistance ← distance/sp;
9          else
10             normDistance ← 1 + distance/maxDistance;
11         end
12         dataset[i, j] ← normDistance;
13     end
14 end
15 return dataset
```

The input of Algorithm 5 is a set of trajectories **T** and a set of *movelets* S'. For each trajectory and each *movelet* it finds the best alignment of the *movelet* over the trajectory, according to Definition 3.3 (line 4). It normalizes the distance value to keep the values between 0 and 2, where the values in the interval $[0, 1]$ are lower or equal than the *movelet* split point and in the interval $(1, 2]$ are greater than the split point (lines 6 to 11). After that, it stores the normalized distance value into the dataset (line 12). It finalizes by returning a dataset in the attribute-value format.

The datasets created in this step are neutral for classification methods, so we can use this step to create first a dataset for a trajectory training set in order to build a classification model, and then a dataset for trajectory test set to evaluate the classification model.

3.3 Complexity Analysis

The complexity of our method is dominated by Algorithm 1. In terms of memory space, it keeps storing three matrices M_1, M_{s-1},

and M_s simultaneously, using $O(n \times m^2)$, where n is the number of trajectories and m is the length of the longest trajectory. Also, its stores at most $O(m \times log\ m)$ candidates for each trajectory. Therefore, the space complexity is $O(n \times m^2)$. In terms of time, the most costly methods are *ComputePointDistances*, *CSD*, and *GenerateCandidates*, that are $O(n \times m^2)$. While the first is performed only n times, the methods *CSD* and *GenerateCandidates* are performed $O(n \times m)$ times. So the overall process requires $O(n^2 \times m^3)$ of time complexity, which is the same complexity of time series shapelets.

4 EXPERIMENTAL SETUP

We evaluate the proposed approach with three different experiments. The first and second experiments are performed over five datasets, among which three are of different trajectory categories (hurricanes, animals, and vehicles). These datasets were used in the experimental evaluation of previous works, so we compare our results with the same datasets in order to make a fair comparison. These experiments show that Movelets is a promising strategy for domain independent trajectory classification problems[2]. The first experiment is a cross-validation evaluation, where we compare Movelets with the results reported by [8] and the methods proposed by [2, 10, 13], that were reimplemented for this paper. This experiment is shown in section 4.1. The second experiment, detailed in section 4.2, is a holdout evaluation, where we compare Movelets with the methods of [2, 10, 13]. This experiment is important for the future reproduction of the results reported in this paper.

The third experiment was performed over the GeoLife dataset [13], and is an evaluation for transportation mode classification, that is one of the most common problems in trajectory classification. As several works for trajectory classification were specifically developed for the transportation mode problem, we show that Movelets is general enough to improve trajectory classification in specific domains. The results of this experiment are detailed in section 4.3.

4.1 Cross-validation Evaluation

In this experiment we use five datasets used in [8], which are described in Table 1 by name, number of trajectories, average length, sampling rate, and class proportion.

Table 1: Datasets description.

	Dataset	# Traj	Length avg (sd)		Classes	
D_1	$\text{Hurricane}_{2,3}$	135	42.11	(17.21)	Scale 2	(46%)
					Scale 3	(64%)
D_2	$\text{Hurricane}_{1,4}$	210	34.84	(17.33)	Scale 1	(71%)
					Scale 4	(29%)
D_3	$\text{Hurricane}_{0,45}$	354	27.44	(17.03)	Scale 0	(76%)
					Scale 4,5	(24%)
D_4	Animals	102	146.96	(62.51)	Elk	(37%)
					Deer	(30%)
					Cattle	(33%)
D_5	Vehicle	421	467.98	(250.53)	Bus	(34%)
					Truck	(66%)

[2]The algorithms and data to reproduce all the experiments are publicly available in https://bitbucket.org/anfer86/acmsac2018_movelets/

Datasets D_1, D_2, and D_3 are related to Atlantic Hurricanes Database collected between 1950 and 2008[3], classified using the Saffir-Simpson scale from 0 to 5, where 0 is the weakest and 5 is the strongest. Dataset D_1 contains the trajectories of scale 2 and 3 hurricanes, D_2 contains the trajectories of scale 1 and 4, and D_3 trajectories of scale 0, 4 and 5.

The dataset D_4 is related to animals movement generated during the Starkey project[4], and contains trajectories of three species observed in June 1995: elk, deer, and cattle. The number of trajectories for each species is 38 (7, 117 points), 30 (4, 333 points), and 34 (3, 540 points), respectively.

The dataset D_5 contains two categories of vehicles moving around the Athens metropolitan area[5]. The trajectories were collected by school buses and trucks, and the number of trajectories for each class is 145 (66, 096 points) and 276 (112, 203 points), respectively.

In this experiment we compare Movelets with (i) the results reported in [8] for the methods RB-TB [5] and TCPR [8]; and (ii) the feature sets proposed by Dodge [2], Zheng [13], and Xiao [10], reimplemented for this paper and publicly available.

To generate the Profile Decomposition proposed by Dodge [2], the threshold value for sinuosity was defined as the mean between minimum and maximum sinuosity values. To fit the best threshold values for the features proposed by Zheng (heading change rate, stop rate and velocity change rate), we generated decision tree models for each feature, as suggested in [13]. For Movelets we only use the spatial dimension and we add the same global features used in [8]: traveled distance, time duration, and average speed.

To build the models we use the algorithms Naive Bayes, C4.5, and SVM, implemented in Weka [9] with default parameters. To evaluate the models we use the weighted average of F-measure (wF), as in [8]. This average consists of the F-measure per class weighted by the class proportion, over a 5-fold cross-validation evaluation. Tables 2, 3, and 4 show the results of cross-validation evaluation for each classifier SVM, C4.5, and Bayes, respectively. The best results for each dataset are highlighted in bold.

As can be seen in the three tables, independently of the classifier (SVM, C4.5, and Bayes), Movelets outperformed almost all existing works in all datasets, loosing only marginally of 0.01 to TCPR with SVM and ending in a tie with TCPR for the classifier C4.5 over the vehicle dataset.

An overall evaluation indicates that from the 15 cases (3 classifiers × 5 datasets) Movelets wins or ties in 14 (93.3%) and looses marginally in only 1 (6.7%). We perform a statistical analysis using the *Friedman's Aligned Rank Test* [3], with level of significance $\alpha = 0.05$, that results in a *p-value* < 0.05. Then, we perform the *post hoc test (control vs. all)* with Movelets as the control and we obtain all *p-values* < 0.05. This result indicates that the probability of the performance differences between Movelets and the other methods has occurred by chance is less than 5%. So, in this experiment our method significantly outperforms the state of the art approaches.

4.2 Holdout Evaluation

The second experiment consists in comparing Movelets with features used in [2, 10, 13], performing a holdout evaluation, dividing

[3]http://weather.unisys.com/hurricane/atlantic/
[4]http://www.fs.fed.us/pnw/starkey/data/tables/
[5]http://www.chorochronos.org/

Table 2: Cross-validation evaluation for SVM.

	SVM					
Dataset	**TB-RB**	**TCPR**	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.46	0.55	0.50	0.50	0.52	**0.75**
$Hurricane_{1,4}$	0.72	0.78	0.72	0.77	0.78	**0.86**
$Hurricane_{0,45}$	0.71	0.87	0.85	0.85	0.86	**0.90**
Animals	0.79	0.89	0.74	0.79	0.87	**0.97**
Vehicle	0.94	**0.99**	0.94	0.84	0.98	0.98

Table 3: Cross-validation evaluation for C4.5.

	C4.5					
Dataset	**TB-RB**	**TCPR**	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.53	0.56	0.47	0.49	0.60	**0.62**
$Hurricane_{1,4}$	0.69	0.73	0.72	0.76	0.74	**0.78**
$Hurricane_{0,45}$	0.71	0.83	0.83	0.83	0.81	**0.85**
Animals	0.80	0.89	0.74	0.83	0.81	**0.96**
Vehicle	0.94	**0.98**	0.85	0.94	0.92	**0.98**

Table 4: Cross-validation evaluation for Bayes.

	Bayes					
Dataset	**TB-RB**	**TCPR**	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.35	0.45	0.53	0.55	0.48	**0.76**
$Hurricane_{1,4}$	0.74	0.79	0.70	0.70	0.66	**0.86**
$Hurricane_{0,45}$	0.71	0.85	0.80	0.82	0.78	**0.87**
Animals	0.70	0.77	0.51	0.70	0.77	**0.91**
Vehicle	0.92	0.97	0.71	0.60	0.74	**0.99**

each dataset into training (60%) and test (40%) sets. We also use the weighted average of F-measure over the test set as the performance metric. Tables 5, 6, and 7, show the results of holdout evaluation for each classifier SVM, C4.5, and Bayes, respectively. The best results for each dataset are highlighted in bold.

For the SVM classifier our method outperforms all state of the art methods in all datasets. For the C4.5 classifier our method outperforms or ties existing works in all datasets, loosing only to the work of Zheng in one dataset. MOVELETS improves the results for dataset D_2 from 0.81 (Dodge) to 0.85, D_4 from 0.76 (Zheng) to 0.93, and D_5 from 0.94 (Xiao) to 0.96. For the Bayes classifier our method significantly outperformed existing approaches in 4 datasets (for D_1 MOVELETS improves the performance from 0.56 (Zheng) to 0.60, for D_2 from 0.72 (Dodge) to 0.80, for D_4 from 0.76 (Xiao) to 0.93, and for D_5 from 0.81 (Xiao) to 0.97). Only for D_3 Zheng achieves 0.86 and MOVELETS 0.84.

An overall evaluation indicates that from the 15 cases (3 classifiers × 5 datasets) MOVELETS wins or ties 13 (86.7%) and marginally looses in 0.02 over 2 datasets (13.3%). We also perform a statistical analysis that results in a *p-value* < 0.05. The *post hoc test (control vs. all)* results in all the *p-values* < 0.05, showing that MOVELETS outperforms the state of the art methods.

Table 5: Holdout evaluation for SVM.

	SVM			
Dataset	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.56	0.59	0.53	**0.60**
$Hurricane_{1,4}$	0.79	0.75	0.77	**0.85**
$Hurricane_{0,45}$	0.87	0.87	0.86	**0.88**
Animals	0.67	0.68	0.81	**0.90**
Vehicle	0.89	0.78	0.98	**0.99**

Table 6: Holdout evaluation for C4.5.

	C4.5			
Dataset	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.39	**0.62**	0.51	**0.62**
$Hurricane_{1,4}$	0.81	0.71	0.76	**0.85**
$Hurricane_{0,45}$	0.84	**0.85**	0.82	0.83
Animals	0.67	0.76	0.74	**0.93**
Vehicle	0.73	0.90	0.94	**0.96**

Table 7: Holdout evaluation for Bayes.

	Bayes			
Dataset	**Dodge**	**Zheng**	**Xiao**	**MOVELETS**
$Hurricane_{2,3}$	0.54	0.56	0.54	**0.60**
$Hurricane_{1,4}$	0.72	0.68	0.68	**0.80**
$Hurricane_{0,45}$	0.76	**0.86**	0.81	0.84
Animals	0.63	0.64	0.76	**0.93**
Vehicle	0.76	0.47	0.81	**0.97**

4.3 Transportation Mode Classification

The works of [2, 10, 13] were specifically developed for transportation mode classification, so the features used in these works solve a specific problem. We compare MOVELETS to these works using also the global features mean and standard deviation of speed and acceleration with *movelets*, since they are basic common features in transportation mode classification. We perform the experiments over the GeoLife database [13], the same used by state-of-the-art methods.

As the original dataset used in [10] was not provided by the authors, we generate a similar size dataset to perform the comparison. From the total of 15, 378 labeled trajectories, we first took the 15, 332 that are of the same six classes used by [10]. Then we removed the 15% shortest and longest trajectories (in terms of number of points). After that, we removed the 15% shortest and longest trajectories (in terms of traveled distance). The remaining 7, 582 trajectories have the following class distribution: *walk* (49.0%), *bus and taxi* (25.0%), *bike* (14.0%), *car* (5.4%), *subway* (5.1%), and *train* (1.5%). As in [10], we used 70% of this dataset for training Random Forest models and 30% to test. Table 8 shows the classification results of F-measure for each class over the testing set for the approaches MOVELETS, Dodge, Zheng, and Xiao. The results show that with MOVELETS

we achieve better results than the other three methods in 5 of 6 transportation modes, and only for transportation mode *walk* Xiao wins with only 0.01.

For a further evaluation, we combine Movelets with the features of Dodge, Zheng, and Xiao, as shown in Table 9. As can be seen, the use of Movelets improved the results of all methods for all classes.

Table 8: Transportation mode classification results.

Class	**Dodge**	**Zheng**	**Xiao**	Movelets
Walk	0.87	0.88	**0.89**	0.88
Bus/Taxi	0.72	0.73	0.76	**0.78**
Bike	0.69	0.72	0.75	**0.78**
Car	0.20	0.29	0.28	**0.60**
Subway	0.36	0.47	0.49	**0.71**
Train	0.28	0.43	0.39	**0.64**

Table 9: Using Movelets with other features.

Class	**Dodge**	**Dodge + Movelets**	**Zheng**	**Zheng + Movelets**	**Xiao**	**Xiao + Movelets**
Walk	0.87	**0.89**	0.88	**0.89**	0.89	**0.90**
Bus/Taxi	0.72	**0.80**	0.73	**0.80**	0.76	**0.82**
Bike	0.69	**0.79**	0.72	**0.80**	0.75	**0.84**
Car	0.20	**0.58**	0.29	**0.59**	0.28	**0.58**
Subway	0.36	**0.70**	0.47	**0.72**	0.49	**0.73**
Train	0.28	**0.62**	0.43	**0.70**	0.39	**0.72**

The results indicate that the better performance for this problem is achieved by the combination of Movelets with the features used by Xiao, as it obtains the better performance for all transportation modes, except for *car*. For *car* trajectories the best result was obtained only with Movelets (see row 4 in Table 8).

We conclude that even though Movelets was not developed for this specific classification problem, it has demonstrated to be very promising as a new general solution for finding relevant subtrajectories for trajectory classification.

5 CONCLUSIONS

In this paper we proposed a new method for extracting relevant subtrajectories for trajectory classification, called Movelets. Our method does neither perform trajectory discretization nor partition, so not loosing important information for class discrimination. The proposed approach is parameter-free and domain independent, what is very important since the parameter definition is a well known problem in data mining, being difficult to estimate and directly affect the data mining results. *Movelets* are very flexible, since they can be combined with any trajectory global or local features without any need of extending or changing the method, so new trajectory features may improve even further movelet based classification. In addition, *movelets* are neutral to classification methods, are easy to visualize, to describe, to understand, and to find them in new trajectory datasets.

Extensive experimental evaluations have demonstrated the effectiveness of the proposed method, that largely outperformed most existing approaches in almost all datasets. On the other hand, although the proposed method does not repeat distance computations to find the *movelets* for each trajectory, the number of distance computations is very high. Therefore, the time complexity is the main drawback of our method. Another problem is that it uses distance functions that are limited to subtrajectories of equal length, in order to compute the distance in linear time. This limitation could be solved by using a time-warp distance, such as DTW, but the complexity would increase to quadratic time.

Future work include the exploration of relevant subtrajectories related to other trajectory dimensions, beyond space; to explore other subtrajectory quality measures; and the looking for ways to reduce the time complexity.

ACKNOWLEDGMENTS

This work was supported by the Brazilian Agency CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

REFERENCES

[1] Lucas Andre de Alencar, Luis Otavio Alvares, Chiara Renso, Alessandra Raffaeta, and Vania Bogorny. 2015. A Rule-based Method for Discovering Trajectory Profiles. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE)*. Pittsburgh, USA, 244–249.

[2] Somayeh Dodge, Robert Weibel, and Ehsan Forootan. 2009. Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects. *Computers, Environment and Urban Systems* 33, 6 (2009), 419–434.

[3] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. 2010. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences* 180, 10 (2010), 2044–2064.

[4] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. 2014. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery* 28, 4 (2014), 851–881.

[5] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. 2008. TraClass: Trajectory Classification Using Hierarchical Region-based and Trajectory-based Clustering. *Proceedings of the Very Large Data Base (VDLB) Endowment* 1, 1 (2008), 1081–1094.

[6] Abdullah Mueen, Eamonn Keogh, and Neal Young. 2011. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. ACM, New York, NY, USA, 1154–1162.

[7] Andrey Tietbohl Palma, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. 2008. A clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the 23rd ACM Symposium On Applied Computing (SAC)*. ACM, Fortaleza, Ceara, Brazil, 863–868.

[8] Dhaval Patel, Chang Sheng, Wynne Hsu, and Mong Li Lee. 2012. Incorporating Duration Information for Trajectory Classification. In *Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE)*. IEEE, Washington, DC, USA, 1132–1143.

[9] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. 2016. *Data Mining: Practical machine learning tools and techniques* (4 ed.). Morgan Kaufmann.

[10] Zhibin Xiao, Yang Wang, Kun Fu, and Fan Wu. 2017. Identifying Different Transportation Modes from Trajectory Data Using Tree-Based Ensemble Classifiers. *ISPRS International Journal of Geo-Information* 6, 2 (2017), 1–22.

[11] Lexiang Ye and Eamonn J. Keogh. 2011. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data Mining Knowledge Discovery* 22, 1-2 (2011), 149–182.

[12] Willian Zalewski, Fabiano Silva, A.G. Maletzke, and C.A. Ferrero. 2016. Exploring Shapelet Transformation for Time Series Classification in Decision Trees. *Knowledge Based Systems* 112, C (Nov. 2016), 80–91.

[13] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. 2010. Understanding transportation modes based on GPS data for web applications. *ACM Transactions on the Web (TWEB)* 4, 1 (2010), 1–36.