View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Publikationsserver der Universität Tübingen

Automatic Construction of Typologies for Massive Collections of Projectile Points

Eamonn Keogh,[1] Lexiang Ye,[1] Taryn Rampley,[2] and Sang-Hee Lee[2]

[1] Department of Computer Science, University of California, Riverside. USA.
[2] Department of Anthropology, University of California, Riverside. USA.

Abstract

In the last few decades there have been several attempts to use computers to automatically construct typologies (keys, classifiers, decision trees, etc) for projectile points. However, all these methods require human effort to extract the features. There are three problems with this. First, it is clearly not scalable to large data collections. Second, human extraction of features is subjective, with all the attendant problems. Finally, and most importantly, all such attempts essentially put the cart before the horse in using preconceived ideas to build classifiers. Here we demonstrate a classification algorithm that tells us the defining features for each class. All the user does is provide two or more labeled sets of photographs/drawings. Our method classifies projectile points with a high degree of accuracy and produces inuitive explanations as to what makes the classes different. We demonstrate the utility of our work on a large dataset of North American projectile points.

Keywords: *projectile points, typology, classification*

1 INTRODUCTION

The classification of projectile points faces a number of problems because there is no universal classificatory system. Work has been informed, instead, by a number of theoretical frameworks that have influenced (1) the choice of attributes used to create classes or types, and (2) the methods that have been applied in segregating classes depending on the questions being asked. Additionally, because data collection of selected projectile point attributes requires human effort, there are limitations to the size of the data set to be analyzed.

We present here preliminary work on the automatic construction of typologies for massive collections of projectile points utilizing time series classification, an approach not previously applied to lithic analysis. In time series classification, the two-dimensional shape outline of the projectile point is converted into a one-dimensional representation called a pseudo-time series. In this case, the time series is an ordered set of n real-valued variables where ordering is not temporal, but spatial.

While the last decade has seen a huge interest in time series classification, to date the most accurate and robust method is the simple nearest neighbor algorithm.[1] algorithm.[1] While the nearest neighbor algorithm has the advantages of simplicity and not requiring extensive parameter tuning, it does have several important disadvantages. Chief among these are its space and time requirements, and the fact that it does not tell us anything about *why* a particular object was assigned to a particular class.

In this work we present a novel time series data mining primitive called *time series shapelets*. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. While we believe shapelets can have many uses in data mining, one obvious implication of them is to mitigate the two weaknesses of the nearest neighbor algorithm noted above.

Because we are defining and solving a new problem, we will take some time to consider a detailed motivating example. Figure 1 shows examples of projectile points from two classes. In order to build a classifier to distinguish between these two types, what features would be appropriate? Due to the effects of resharpening, size is not always a reliable method of classification. However, shape is a generally accepted method for classifying projectile points. As illustrated in figure 1, the overall shape of the projectile points is fairly similar. Both types may be described as triangular, side notched, eared projectile points with straight to concave bases. Furthermore, it is not uncommon to recover broken projectile points which are likely to confuse any global measures of shape.

[1]Hui Ding et al., "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," *Proceedings of the 34th International Conference on Very Large Data Bases* (2008) 1542–1552; Steven L. Salzberg, "On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach," *Data Mining and Knowledge Discovery* 1 (1997): 317–328; Xiaopeng Xi et al., "Fast Time Series Classification Using Numerosity Reduction," *Proceedings of the 23rd International Conference on Machine Learning* 148 (2006) 1033–1040.

Instead, we attempt the following. We first convert each projectile point into a one-dimensional representation as shown in figure 2.

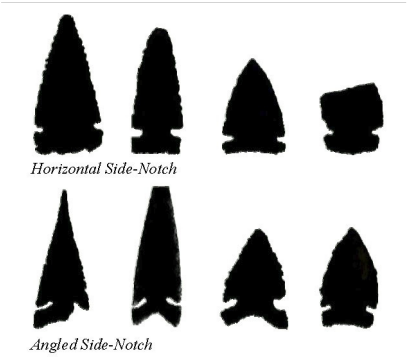

Figure 1. *Examples of projectile points from two classes. Note that several points have damage due to breakage.*

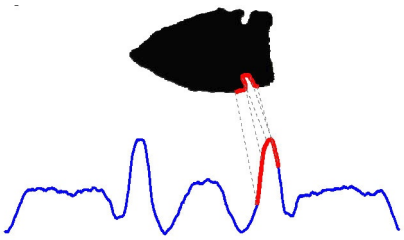

Figure 2. *A shape can be converted into a one dimensional "time series" representation. The reason for the highlighted section of the time series will be made apparent shortly.*

Such representations have been successfully used for the classification, clustering, and outlier detection of shapes in recent years.[1] However, here we find that using a nearest neighbor classifier with either the (rotation invariant) Euclidean distance or Dynamic Time Warping (DTW) distance does not significantly outperform random guessing. The reason for the poor performance of these otherwise very competitive classifiers seems to be due to the fact that the data is somewhat noisy (due to breakage, resharpening, etc.), and this noise is enough to swamp the subtle differences in the shapes.

Suppose, however, that instead of comparing the *entire* shapes, we only compare a *small* subsection of the shapes from the two classes that is particularly discriminating. We can call such subsections *shapelets*, which invokes the idea of a small "sub-shape" (see fig. 3). For the moment we ignore the details of how to formally define shapelets, and how to efficiently compute them.

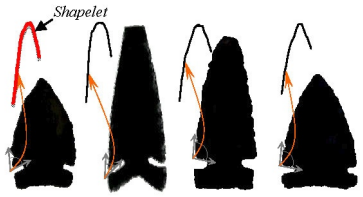

Figure 3. *Here, the shapelet hinted at in figure 2 (in both cases shown with a bold line) is the subsequence that best discriminates between the two classes.*

As we can see, the shapelet has "discovered" that the defining difference between the two classes of projectile points is that points with angled side-notches have barb tips that are defined by a more acute angle. Having found the shapelet and recorded its distance to the nearest matching subsequence in all objects in the database, we can build the simple decision-tree classifier shown in figure 4.

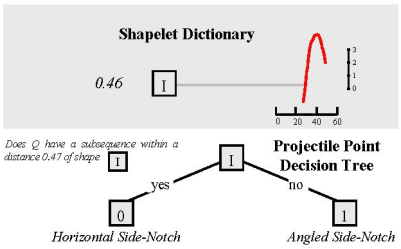

Figure 4. *A decision-tree classifier for the projectile point problem. The object to be classified has all of its subsequences compared to the shapelet, and if any subsequence is less than (the empirically determined value of) 0.47, it is classified as* Horizontal Side-Notch.

The reader will immediately see that this method of classification has many potential advantages over current methods:

- Shapelets can provide interpretable results, which may help domain practitioners better understand their data. Most other state-of-the-art time series/shape classifiers do not produce interpretable results.[2]
- Shapelets can be significantly more accurate/robust on some datasets. This is because they are local features, whereas most other state-of-the-art time series/shape classifiers consider global features, which can be brittle to even low levels of noise and

[1]Eamonn Keogh et al., "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006) 882–893.

[2]Hui Ding et al., "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," *Proceedings of the 34th International Conference on Very Large Data Bases* (2008) 1542–1552; Eamonn Keogh and Shruti Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Proceedings of the 8th ACM Special Interest Group on Knowledge Discovery and Data Mining* (2005) 102–111.

distortions.[5] In our example, projectile points which have damage are still usually correctly classified.

- Shapelets can be significantly faster at classification than existing state-of-the-art approaches. The classification time is just O(*ml*), where *m* is the length of the query time series and *l* is the length of the shapelet. In contrast, if we use the best performing global distance measure, rotation invariant DTW distance,[1] the time complexity is on the order of O(km^3), where *k* is the number of reference objects in the training set.[2] On real-world problems the speed difference can be greater than three orders of magnitude.

2 RELATED WORK AND BACKGROUND

While there is a vast amount of literature on time series classification and mining,[3] we believe that the problem we intend to solve here is unique. The closest work is that of Geurts.[4] Here the author also attempts to find local patterns in a time series which are predictive of a class. However, the author considers the problem of finding the *best* such pattern intractable, and thus resorts to examining a single, randomly chosen instance from each class, and even then only considering a reduced piecewise constant approximation of the data. While the author notes "it is impossible in practice to consider every such subsignal as a candidate pattern," this is in fact *exactly* what we do, aided by eight years of improvements in CPU time, and, more importantly, an admissible pruning technique that can prune off more than 99.9% of the calculations. Our work may also be seen as a form of *supervised* motif discovery algorithm.[5]

2.1 Traditional Projectile Point Classification

The classification of projectile points is a complex problem because there is no universal classificatory system and work has been informed by a number of theoretical frameworks that have influenced the choice of attributes used to create classes or types. In addition, the methods that have been applied in segregating classes have varied depending on the questions being asked. Much of the early work in projectile point classification was concerned with the construction of culture-history and was largely descriptive.[6] Later work was concerned with explaining culture change, and more recently, an interest in evolutionary archaeology has applied a biological perspective to understanding changes in artifact form.[7]

Traditional methods of projectile point classification can only employ a limited number from a nearly infinite number of attributes to create types or classes. Commonly used non-metric attributes include flaking patterns on the face of the point, haft characteristics, notch and base shapes, and edge treatments. Early work in descriptive classification called for consistent terminology so that readers could develop "…a proper mental picture of the object."[8] Classification was accomplished by description of the geometric shape of the point and how the point was modified. Finkelstein[9] suggested a taxonomic system with the goal of providing an objective means of forming purely morphological classes.

---

[1]Eamonn Keogh et al., "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006), 882–893.

[2]There are techniques to mitigate the cubic complexity of rotation invariant DTW, but unlike shapelets, the time *is* dependent on **D.**

[3]Eamonn Keogh and Shruti Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration,"*Proceedings of the 8th ACM Special Interest Group on Knowledge Discovery and Data Mining* (2005): 102–111; Hui Ding et al., "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," *Proceedings of the 34th International Conference on Very Large Data Bases (2008): 1542–1552;* Xiaopeng Xi et al., "Fast Time Series Classification Using Numerosity Reduction," *Proceedings of the 23rd International Conference on Machine Learning* (2006): 1033–1040.

[4]Pierre Geurts, "Pattern Extraction for Time Series Classification," *Proceedings of the 5th Conference on Principles and Practice of Knowledge Discovery in Databases* (2005) 115–127.

[5]Bill Chiu et al., "Probabilistic Discovery of Time Series Motifs," *Proceedings of the 9th Conference on Principles and Practice of Knowledge Discovery in Databases* (2003) 493–498.

[6]Glenn A. Black and Paul Weer, "A Proposed Terminology for Shape Classifications of Artifacts," *American Antiquity* 1 (1936): 280–294; Joe Finkelstein, "A Suggested Projectile-Point Classification," *American Antiquity* 2 (1937): 197–203.

[7]Briggs Buchanan and Mark Collard, "Investigating the Peopling of North America through Cladistic Analyses of Early Paleoindian Projectile Points," *Journal of Anthropological Archaeology* 26 (2007): 366–393; Briggs Buchanan and Mark Collard, "Phenetics, Cladistics, and the Search for the Alaskan Ancestors of the Paleoindians: A Reassessment of Relationships among the Clovis, Nenana, and Denali Archaeological Complexes," *Journal of Archaeological Science* 36 (2008): 1683–1694; Michael J. O'Brien et al., "Cladistics Is Useful for Reconstructing Archaeological Phylogenies: Paleoindian Points from the Southeastern United States," *Journal of Archaeological Science* 28 (2001): 1115–1136; Michael J. O'Brien et al., "Two Issues in Archaeological Phylogenetics: Taxon Construction and Outgroup Selection," *Journal of Theoretical Biology* 215 (2002): 133–150.

[8]Glenn A. Black and Paul Weer, "A Proposed Terminology for Shape Classifications of Artifacts," *American Antiquity* 1 (1936): 280–294.

[9]J. Joe Finkelstein, "A Suggested Projectile-Point Classification," *American Antiquity* 2 (1937): 197–203.

With an interest in increasing objectivity and the ability to more easily apply complex statistical analyses, metric attributes have also gained importance, including length, width, thickness, and weight, the ratios of which may also be used to summarize the overall "shape" of a projectile point.[1]

While each of these methods has demonstrated utility in being able to discriminate different classes of projectile points, they also suffer from some disadvantages. All of these methods, even those relying heavily on computers for complex statistical analysis, require human effort to extract features or measurements, making these methods unsuitable for very large datasets because of the time involved in data extraction. Additionally, human extraction is subjective and susceptible to inter-observer error.

The traditional approach poses a problem in data-sharing and the comparability of results. A different approach, which has not previously been applied to lithic analysis, is the conversion of the two-dimensional shape outline of the projectile point to a one-dimensional representation called a pseudo-time series (referred to here as time series).

2.2 Converting Shapes into Time Series

Computer technologies have been used to identify or classify shapes in various domains, such as bone fragments, projectile points, and petroglyphs. To make the similarity measure invariant to many distortions like scale and offset, we use a well-known method to convert the shape into time series. The example of obtaining a time series from a projectile point is shown in figure 5.

Using this method, we simply convert the two-dimensional image into a one-dimensional representation. Note that this method is only one of many proposed in the literature. A recent paper[2] has shown that this method achieves higher or at least the same accuracy as six other "sophisticated" methods. Therefore, we prefer this method because of its simplicity.

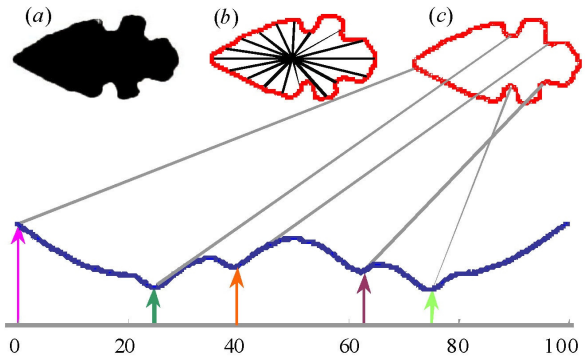

Figure 5. *Convert projectile point to time series. (a) An original photo of projectile point. (b) The value of each point of the projectile point shape is represented by the distance between the point itself and the center; see (c), where the X-axis represents continuous points of the shape and the Y-axis is the distance value of the point.*

2.3 Notation

Symbol	Explanation
T, R	time series
S	subsequence
m, $\|T\|$	length of time series
l, $\|S\|$	length of subsequence
d	distance measurement
D	time series dataset
A,B	class label
I	entropy
$\hat{I}$	weighted average entropy
sp	split strategy
k	number of time series objects in
C	classifier
$S_{(k)}$	the kth data point in subsequence S

Table 1. *Symbol table.*

Table 1 summarizes the notation in the paper; we expand on the definitions below. We begin by defining the key terms in the paper. For ease of exposition, we consider only a two-class problem. However, extensions to a multiple-class problem are trivial.

Definition 1: *Time Series*. A time series $T = t_1,\ldots,t_m$ is an ordered set of m real-valued variables.

Data points $t_1,\ldots,t_m$ are typically arranged by temporal order, spaced at equal time intervals. We are interested in the *local* properties of a time series rather than the *global* properties. A local subsection of time series is termed as a *subsequence*.

[1]Briggs Buchanan and Mark Collard, "Investigating the Peopling of North America through Cladistic Analyses of Early Paleoindian Projectile Points,"*Journal of Anthropological Archaeology* 26 (2007): 366–393; Briggs Buchanan, "An Analysis of Folsom Projectile Point Re-sharpening Using Quantitative Comparisons of Form and Allometry," *Journal of Archaeological Science* 33 (2006): 185–199; Michael J. Shott, "Stones and Shafts Redux: The Metric Discrimination of Chipped-Stone Dart and Arrow Points," *American Antiquity* 62 (1997): 86–101; David H. Thomas, "Arrowheads and Atlatl Darts: How the Stones Got the Shaft," *American Antiquity* 43 (1978): 461–472.

[2]Eamonn Keogh et al., "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006) 882–893.

Definition 2: *Subsequence*. Given a time series T of length m, a subsequence S of T is a sampling of length $l \leq m$ of contiguous positions from T, that is, $S = t_p,\ldots,t_{p+l-1}$, for $1 \leq p \leq m - l + 1$.

Our algorithm needs to extract all of the subsequences of a certain length. This is achieved by using a sliding window of the appropriate size.

Definition 3: *Sliding Window*. Given a time series T of length m, and a user-defined subsequence length of l, all possible subsequences can be extracted by sliding a window of size l across T and considering each subsequence S_p^l of T. Here the superscript l is the length of the subsequence and subscript p indicates the starting position of the sliding window in the time series. The set of all subsequences of length l extracted from T is defined as $\mathbf{S}_T^l$, $\mathbf{S}_T^l = \{S_p^l$ of T, for $1 \leq p \leq m - l + 1\}$.

As with virtually all time series data mining tasks, we need to provide a similarity measure between the time series $Dist(T, R)$.

Definition 4: *Distance between the time series*. $Dist(T, R)$ is a distance function that takes two time series T and R which are of the same length as inputs and returns a nonnegative value d, which is said to be the distance between T and R. We require that the function $Dist$ be symmetrical; that is, $Dist(R, T) = Dist(T, R)$.

The $Dist$ function can also be used to measure the distance between two subsequences of the same length, since the subsequences are of the same format as the time series. However, we will also need to measure the similarity between a short subsequence and a (potentially much) longer time series. We therefore define the distance between two time series T and S, with $|S| < |T|$ as:

Definition 5: *Distance from the time series to the subsequence*. $SubsequenceDist(T, S)$ is a distance function that takes time series T and subsequence S as inputs and returns a nonnegative value d, which is the distance from T to S. $SubsequenceDist(T, S) = \min(Dist(S, S'))$, for $S' \in \mathbf{S}_T^{|S|}$.

Intuitively, this distance is simply the distance between S and its best matching location somewhere in T, as shown in figure 6.

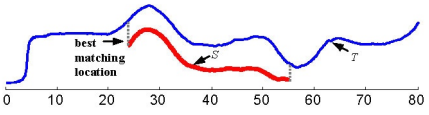

Figure 6. *Illustration of best matching location in time series T for subsequence S.*

As we shall explain in Section 3, our algorithm needs some metric to evaluate how well it can divide the entire combined dataset into two original classes. Here, we use concepts very similar to the *information gain* used in the traditional decision tree.[1] The reader may recall the original definition of entropy which we review here:

Definition 6: *Entropy*. A time series dataset **D** consists of two classes, A and B. Given that the proportion of objects in class A is $p(A)$ and the proportion of objects in class B is $p(B)$, the entropy of **D** is: $I(\mathbf{D}) = -p(A)\log(p(A)) - p(B)\log(p(B))$.

Each splitting strategy divides the whole dataset **D** into two subsets, $\mathbf{D_1}$ and $\mathbf{D_2}$. Therefore, the information remaining in the entire dataset after splitting is defined by the weighted average entropy of each subset. If the fraction of objects in $\mathbf{D_1}$ is $f(\mathbf{D_1})$ and the fraction of objects in $\mathbf{D_2}$ is $f(\mathbf{D_2})$, the total entropy of **D** after splitting is $\hat{I}(\mathbf{D}) = f(\mathbf{D_1})I(\mathbf{D_1}) + f(\mathbf{D_2})I(\mathbf{D_2})$. This allows us to define the information gain for any splitting strategy:

Definition 7: *Information Gain*. Given a certain split strategy sp which divides **D** into two subsets $\mathbf{D_1}$ and $\mathbf{D_2}$, the entropy before and after splitting is $I(\mathbf{D})$ and $\hat{I}(\mathbf{D})$. So the information gain for this splitting rule is

$$Gain(sp) = I(\mathbf{D}) - \hat{I}(\mathbf{D}),$$
$$Gain(sp) = I(\mathbf{D}) - f(\mathbf{D_1})I(\mathbf{D_1}) + f(\mathbf{D_2})I(\mathbf{D_2}).$$

As hinted at in the introduction, we use the distance to a *shapelet* as the splitting rule. The shapelet is a subsequence of a time series such that most of the time series objects in one class of the dataset are close to the shapelet under *SubsequenceDist,* while most of the time series objects from the other class are far away from it.

To find the best shapelet, we may have to test many shapelet candidates. In the brute force algorithm discussed in Section 3.1, given a candidate shapelet, we calculate the distance between the candidate and every time series object in the dataset. We sort the objects according to the distances and find an optimal split point between two neighboring distances.

Definition 8: *Optimal Split Point* (*OSP*). A time series dataset **D** consists of two classes, A and B. For a shapelet candidate S, we choose some distance threshold d_{th} and split **D** into $\mathbf{D_1}$ and $\mathbf{D_2}$, such that for every time series object $T_{1,i}$ in $\mathbf{D_1}$, $SubsequenceDist(T_{1,i}, S) < d_{th}$ and for every time series object $T_{2,i}$ in $\mathbf{D_2}$, $SubsequenceDist(T_{2,i}, S) \geq d_{th}$. An *Optimal Split Point* is a distance threshold that

$$Gain(S, d_{OSP(\mathbf{D}, S)}) \geq Gain(S, d'_{th})$$

for any other distance threshold d'_{th}.

So using the shapelet, the splitting strategy contains two factors: the shapelet and the corresponding optimal split point. As a concrete example, in figure 4 the shapelet is shown in red in the shapelet dictionary, and the optimal split point is 0.47.

We are finally in the position to formally define the shapelet.

[1] Leo Breiman et al., *Classification and Regression Trees* (Belmont, CA: Wadsworth, 1984).

Definition 9: *Shapelet*. Given a time series dataset **D** which consists of two classes, *A* and *B*, *shapelet*(**D**) is a subsequence that, with its corresponding optimal split point, $Gain(shapelet(\mathbf{D}), d_{OSP(\mathbf{D}, shapelet(\mathbf{D}))}) \geq Gain(S, d_{OSP(\mathbf{D}, S)})$ for any other subsequence *S*.

Since the shapelet is simply *any* time series of some length less than or equal to the length of the shortest time series in our dataset, there are an infinite amount of possible shapes it could have. For simplicity, we assume the shapelet to be a subsequence of a time series object in the dataset. It is reasonable to make this assumption since the time series objects in one class presumably contain some similar subsequences, and these subsequences are good candidates for the shapelet.

Nevertheless, there is still a very large number of possible shapelet candidates. Suppose the dataset **D** contains *k* time series objects. We specify the minimum and maximum length of the shapelet candidates that can be generated from this dataset as *MINLEN* and *MAXLEN*, respectively. Obviously $MAXLEN \leq \min(m_i)$, m_i is the length of the time series T_i from the dataset, $1 \leq i \leq k$. Considering a certain fixed length *l*, the number of shapelet candidates generated from the dataset is:

$$\sum_{T_i \in D} (m_i - l + 1)$$

So the *total* number of candidates of all possible lengths is:

$$\sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (m_i - l + 1)$$

If the shapelet can be any length smaller than that of the shortest time series object in the dataset, the number of shapelet candidates is linear in *k*, and quadratic in (the average $\overline{m}$ length of time series objects). For example, the three-class Avonlea-Clovis-Mixture projectile point dataset we will see in Section 5 has 36 instances, each of length 1030. If we set *MINLEN*=3, *MAXLEN*=1030, there will be 19,040,616 shapelet candidates. For each of these candidates, we need to find its nearest neighbors within the *k* time series objects. Using the brute force search, it will take approximately four days to accomplish this. However, as we will show in Section 3, we can achieve an identical result in a tiny fraction of this time with a novel pruning strategy.

3 FINDING THE SHAPELET

We first show the brute force algorithm for finding shapelets, followed by two simple but highly effective speedup methods.

3.1 Brute Force Algorithm

The most straightforward way for finding the shapelet is using the brute force method. The algorithm is described in table 2.

	FindingShapeletBF (dataset **D**, *MAXLEN*, *MINLEN*)
1	*candidates* ← GenerateCandidates(**D**, *MAXLEN*, *MINLEN*)
2	*bsf_gain* ← 0
3	**For each** *S* **in** *candidates*
4	*gain* ← CheckCandidate(**D**, *S*)
5	**If** *gain* > *bsf_gain*
6	*bsf_gain* ← *gain*
7	*bsf_shapelet* ← *S*
8	**EndIf**
9	**EndFor**
10	**Return** *bsf_shapelet*

Table 2. *Brute force algorithm for finding shapelet.*

Given a combined dataset **D**, in which each time series object is labeled either class *A* or class *B*, along with the user-defined maximum and minimum lengths of the shapelet, line 1 generates all of the subsequences of all possible lengths, and stores them in the unordered list *candidates*. After initializing the best information gain *bsf_gain* to be zero (line 2), the algorithm checks how well each candidate in *candidates* can separate objects in class *A* and class *B* (lines 3 to 7). For each shapelet candidate, the algorithm calls the function CheckCandidate() to obtain the information gain achieved if using that candidate to separate the data (line 4). As illustrated in figure 7, we can visualize this as placing class-annotated points on the real number line, representing the distance of each time series to the candidate. Intuitively, we hope to find that this mapping produces two well-separated "pure" groups. In this regard the example in figure 7 is very good, but clearly not perfect.

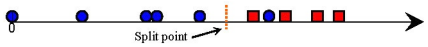

Figure 7. *The CheckCandidate() function at the heart of the brute force algorithm can be regarded as testing to see how mapping all of the time series objects on the number line based on their SubsequenceDist(T, S) separates the two classes.*

If the information gain is higher than the *bsf_gain*, the algorithm updates the *bsf_gain* and the corresponding best shapelet candidate *bsf_shapelet* (lines 5 to 7). Finally, the algorithm returns the candidate with the highest information gain in line 10. The two subroutines GenerateCandidates() and CheckCandidate() called in the algorithm are outlined in
Table 3 and
Table 4, respectively.
In
Table 3, the algorithm GenerateCandidates() begins by initializing the shapelet candidate pool to be an empty set and the shapelet length *l* to be *MAXLEN* (lines 1 and 2).

Thereafter, for each possible length *l*, the algorithm slides a window of size *l* across all of the time series objects in the dataset **D**, extracts all of the possible candidates and adds them to the *pool* (line 5). The algorithm finally returns the *pool* as the entire set of shapelet candidates that we are going to check (line 9).

In table 4 we show how the algorithm evaluates the utility of each candidate by using the information gain.

GenerateCandidates (dataset **D**, *MAXLEN*, *MINLEN*)	
1	*pool* ← Ø
2	*l* ← *MAXLEN*
3	**While** *l* ≥ *MINLEN*
4	**For** *T* **in D**
5	*pool* ← *pool* ∪ $\mathbf{S}_T^l$
6	**EndFor**
7	*l* ← *l*-1
8	**EndWhile**
9	**Return** *pool*

Table 3. *Generate all the candidates from time series dataset.*

CheckCandidate (dataset **D**, shapelet candidate *S*)	
1	*objects_histogram* ← Ø
2	**For each** *T* **in D**
3	*dist* ← SubsequenceDist(*T*, *S*)
4	insert *T* into *objects_histogram* by the key *dist*
5	**EndFor**
6	**Return** CalculateInformationGain(*objects_histogram*)

Table 4. *Checking the utility of a single candidate.*

First, the algorithm inserts all of the time series objects into the histogram *objects_histogram* according to the distance from the time series object to the candidate in lines 1 to 4. After that, the algorithm returns the utility of that candidate by calling CalculateInformationGain() (line 6).

The CalculateInformationGain() subroutine, as shown in Table 5, takes an object histogram as the input, finds an optimal split point *split_dist* (line 1) and divides the time series objects into two subsets by comparing the distance to the candidate with *split_dist* (lines 4 to 7). Finally, it calculates the information gain (cf. definitions 6, 7) of the partition and returns the value (line 10).

CalculateInformationGain (distance histogram *obj_hist*)	
1	*split_dist* ← OptimalSplitPoint(*obj_hist*)
2	$\mathbf{D_1}$ ← Ø, $\mathbf{D_2}$ ← Ø
3	**For** *d* **in** *obj_hist*
4	**If** *d.dist* < *split_dist*
5	$\mathbf{D_1}$ ← $\mathbf{D_1}$ ∪ *d.objects*
6	**Else**
7	$\mathbf{D_2}$ ← $\mathbf{D_2}$ ∪ *d.objects*
8	**EndIf**
9	**EndFor**
10	**Return** $I(\mathbf{D})$-$\hat{I}(\mathbf{D})$

Table 5. *Information gain of distance histogram optimal split.*

After building the distance histogram for all of the time series objects to a certain candidate, the algorithm will find a split point that divides the time series objects into two subsets (denoted by the dashed line in figure 7). As noted in definition 8, an optimal split point is a distance threshold. Comparing the distance from each time series object in the dataset to the shapelet with the threshold, we can divide the dataset into two subsets, which achieves the highest information gain among all of the possible partitions. Any point on the positive real number line could be a split point, so there are infinite possibilities from which to choose. To make the search space smaller, we check only the mean values of each pair of adjacent points in the histogram as a possible split point. This reduction still finds all of the possible information gain values, since the information gain cannot change in the region *between* two adjacent points. Furthermore, in this way, we maximize the margin between two subsets.

The naïve brute force algorithm clearly finds the optimal shapelet. It appears that it is extremely space inefficient, requiring the storage of all of the shapelet candidates. However, we can mitigate this with some internal bookkeeping that generates and then discards the candidates one at a time. Nevertheless, the algorithm suffers from high time complexity. Recall that the number of the time series objects in the dataset is k and the average length of each time series is $\overline{m}$. As we discussed in Section 2.3, the size of the candidate set is $O(\overline{m}^2 k)$. Checking the utility of one candidate takes $O(\overline{m}k)$. Hence, the overall time complexity of the algorithm is $O(\overline{m}^3 k^2)$, which makes the real-world problems intractable.

3.2 Subsequence Distance Early Abandon

In the brute force method, the distance from the time series T to the subsequence S is obtained by calculating the Euclidean distance of every subsequence of length $|S|$ in T and S and choosing the minimum. This takes $|T|$-$|S|$+1 distance calculations between subsequences. However, all we need to know is the *minimum* distance rather than all of the distances. Therefore, instead of calculating the exact distance between every subsequence and the candidate, we can stop distance calculations once the partial distance exceeds the minimum distance known so far. This trick is known as *early abandon*,[1] which is very simple yet has been shown to be extremely effective for similar types of problems.

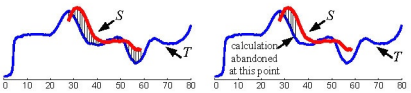

Figure 8. (left) *Illustration of complete Euclidean distance.* (right) *Illustration of Euclidean distance early abandon.*

While it is a simple idea, for clarity we illustrate the idea in figure 8 and provide the pseudo code in table 6.

[1]Eamonn Keogh et al., "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006) 882–893.

```
SubsequenceDistanceEarlyAbandon(T, S)
1   min_dist ← ∞
2   stop ← False
3   For S_i in S_T^|S|
4     sum_dist ← 0
5     For k ← 1 to |S|
6       sum_dist ← sum_dist + (S_i(k) − S_(k))^2
7       If sum_dist ≥ min_dist
8         stop ← True
9         Break
10      EndIf
11    EndFor
12    If not stop
13      min_dist ← sum_dist
14    EndIf
15  EndFor
16  Return min_dist
```

Table 6. *Early abandon the non-minimum distance.*

In line 1, we initialize the minimum distance *min_dist* from the time series T to the subsequence S to be infinity. Thereafter, for each subsequence S_i from T of length $|S|$, we accumulate the distance *sum_dist* between S_i and S, one data point at a time (line 6). Once *sum_dist* is larger than or equal to the minimum distance known so far, we abandon the distance calculation between S_i and S (lines 7 to 9). If the distance calculation between S_i and S finishes, we know that the distance is smaller than the minimum distance known so far. Thus, we update the minimum distance *min_dist* in line 13. The algorithm returns the true distance from the time series T to the subsequence S in line 16. As we will demonstrate later, this simple trick reduces the time required by a large, constant factor.

3.3 Admissible Entropy Pruning

Our definition of the shapelet requires some measure of how well the distances to a given time series subsequence can split the data into two "purer" subsets. The reader will recall that we used the information gain (or entropy) as that measure. However, there are other commonly used measures for distribution evaluation, such as the Wilcoxon signed-rank test.[1] We adopted the entropy evaluation for two reasons. First, it is easily generalized to the multi-class problem. Second, as we will now show, we can use a novel idea called *early entropy pruning* to avoid a large fraction of distance calculations required when finding the shapelet.

Obtaining the distance between a candidate and its nearest matching subsequence of each of the objects in the dataset is the most expensive calculation in the brute force algorithm, whereas the information gain calculation takes an inconsequential amount of time. Based on this observation, instead of waiting until we have all of the distances from each of the time series objects to the candidate, we can calculate an *upper bound* of the information gain based on the currently observed distances. If at any point during the search the upper bound cannot beat the best-so-far information gain, we stop the distance calculations and prune that particular candidate from consideration, secure in the knowledge that it cannot be a better candidate than the current best so far.

In order to help the reader understand the idea of pruning with an upper bound of the information gain, we consider a simple example. Suppose, as shown in figure 9, ten time series objects are arranged in a one-dimensional representation by measuring their distance to the best-so-far candidate. This happens to be a good case, with five of the six objects from class *A* (represented by circles) closer to the candidate than any of the four objects from class *B* (represented by squares). In addition, of the five objects to the right of the split point, only one object from class *A* is mixed up with the class *B*. The optimal split point is represented by a vertical dashed line, and the best-so-far information gain is:

Figure 9. *Distance arrangement of the time series objects in a one-dimensional representation of best-so-far information gain. The positions of the objects represent their distances to the candidate.*

We now consider another candidate. The distances of the first five time series objects to the candidate have been calculated, and their corresponding positions in a one-dimensional representation are shown in figure 10.

Figure 10. *The arrangement of the first five distances from the time series objects to the candidate.*

We can ask the following question: of the 30,240 distinct ways the remaining five distances could be added to this line, could any of them result in an information gain that is better than the best so far? In fact, we can answer this question in constant time. The idea is to imagine the most optimistic scenarios and test them. It is clear that there are only two optimistic possibilities: either all of the remaining class *A* objects map to the far right and all of the class *B* objects map to the far left, or vice versa. Figure 11 shows the former scenario applied to the example shown in figure 10.

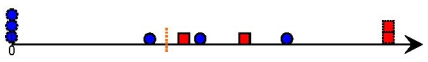

Figure 11. *One optimistic prediction of a distance distribution based on distances that have already been calculated in fig. 10. The dashed objects are in the optimistically assumed placements.*

The information gain of the better of the two optimistic predictions is:

[1]Frank Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics* 1 (1945): 80–83.

[-(6/10)log(6/10)-(4/10)log(4/10)]-[(4/10)[-(4/4)log(4/4)]+(6/10)[-(4/6)log(4/6)-(2/6)log(2/6)]]=0.2911

which is *lower* than the best-so-far information gain. Therefore, at this point, we can stop the distance calculation for the remaining objects and prune this candidate from consideration forever. In this case, we saved 50% of the distance calculations. But in real-life situations, early entropy pruning is generally much more efficient than we have shown in this brief example. This intuitive idea is formalized in the algorithm outlined in table 7. The algorithm takes as the inputs the best-so-far information gain, the calculated distances from objects to the candidate organized in the histogram (i.e., the number line for figures 8, 9 and 10) and the remaining time series objects in class *A* and class *B*, and returns TRUE if we can prune the candidate as the answer. The algorithm begins by finding the two ends of the histogram (discussed in Section 3.1). For simplicity, we make the distance values at two ends as 0 and maximum distance +1 (in lines 1 and 2). To build the optimistic histogram of the whole dataset based on the existing one (lines 3 and 8), we assign the remaining objects of one class to one end and those of the other class to the other end (lines 4 and 9). If in either case, the information gain of the optimistic histogram is higher than the best-so-far information gain (lines 5 and 10), it is still possible that the actual information gain of the candidate can beat the best so far. Thus, we cannot prune the candidate and we should continue to test (lines 6 and 11). Otherwise, if the upper bound of the actual information gain is lower than the best so far, we save all of the remaining distance calculations with this candidate (line 13).

The utility of this pruning method depends on the data. If there is any class-correlated structure in the data, we will typically find a good candidate that gives a high information gain early in our search, and thereafter the vast majority of candidates will be pruned quickly.

	EntropyEarlyPrune (*bsf_gain*, *dist_hist*, c_A, c_B)
	minend ← 0
	maxend ← largest distance value in *dist_hist* + 1
1	*pred_dist_hist* ← *dist_hist*
2	Add to the *pred_dist_hist*, c_A at *minend* and c_B at
3	*maxend*
4	**If** CalculateInformationGain (*pred_dist_hist*) >
5	*bsf_gain*
6	**Return FALSE**
7	**EndIf**
8	*pred_dist_hist* ← *dist_hist*
9	Add to the *pred_dist_hist*, c_A at *maxend* and c_B at
10	*minend*
11	**If** CalculateInformationGain (*pred_dist_hist*) >
12	*bsf_gain*
13	**Return FALSE**
	EndIf
	Return TRUE

Table 7. *Information gain upper bound pruning.*

There is one simple trick we can do to get the maximum pruning benefit. Suppose we tested all of the objects from class *A* first, then all of the objects from class *B*. In this case, the upper bound of the information gain must always be maximum until at least after the point at which we have seen the first object from class *B*. We therefore use a round-robin algorithm to pick the next object to be tested. That is to say, the ordering of objects we use is a_1, b_1, a_2, b_2,…, a_n, b_n. This ordering lets the algorithm know very early in the search if a candidate cannot beat the best so far.

3.4 Techniques for Breaking Ties

It is often the case that different candidates will have the same best information gain. This is particularly true for small datasets. We propose several options to break this tie, depending on the application. We can break such ties by favoring the longest candidate, the shortest candidate, or the one that achieves the largest margin between the two classes.

3.4.1 *Longest Candidate*

The longest candidate always contains the entire distinguishing feature that is in one class and absent from the other class. However, it is possible that the longest candidate might also contain some irrelevant information or noise near the distinguishing feature subsequence, which is likely to reduce the accuracy in some applications.

3.4.2 *Shortest Candidate*

In contrast, favoring the shortest candidate can avoid noise in the shapelet. The shortest candidate is useful when there are multiple, discontinuous features in one class. To enumerate each of these shapelets, each time, after the algorithm finds a shapelet, we replace the feature subsequences in all of the time series objects with random walk subsequences of the same length and rerun the algorithm. By running the algorithm multiple times like this, the method will return short, discontinuous shapelets.

3.4.3 *Maximum Separation*

Using the maximum separation to break the tie follows the same basic idea of SVMs. The algorithm finds the shapelet that maximizes the distance between the two classes. The distance between two classes is defined by first calculating the mean distances from the time series objects in each individual class to the shapelet, then returning the difference between the mean distances.

Based on comprehensive experiments, the best accuracy is most often achieved when breaking the tie using the shapelet that has the maximum margin, which is reasonable since this rule maximizes the difference between classes.

4 SHAPELETS FOR CLASSIFICATION

While we believe that shapelets can have implications for many time series data mining problems, including visualization, anomaly detection, and rule discovery, for

brevity we will focus only on the classification problem in this work. Classifying with a shapelet and its corresponding split point produces a binary decision as to whether a time series belongs to a certain class or not. Obviously, this is not enough to deal with a multi-class situation. Even with two-class problems, a linear classifier is sometimes inadequate. In order to make the shapelet classifier universal, we frame it as a decision tree.[1] Given the discussion of the information gain above, this is a natural fit. At each step of the decision tree induction, we determine the shapelet and the corresponding split point over the training subset considered in that step. (A similar idea is considered in Geurts[2]).

After the learning procedure finishes, we can assess the performance of the shapelet decision tree classifier by calculating the accuracy on the testing dataset. The way we predict the class label of each testing time series object is very similar to the way this is done with a traditional decision tree. For concreteness the algorithm is described in table 8.

CalculateAccuracy (shapelet decision tree classifier **C**, dataset $\mathbf{D}_t$)	
1	**For each** T **in** $\mathbf{D}_t$
2	predict_class_label ← Predict(**C**, T)
3	**If** predict_class_label is the same as actual class label
4	correct ← correct + 1
5	**EndIf**
6	**EndFor**
7	**Return** correct / | $\mathbf{D}_t$ |

Table 8. *Calculating the accuracy of the shapelet classifier.*

The technique to predict the class label of each testing object is described in table 9. For each node of the decision tree, we have the information of a single shapelet classifier, the left subtree and the right subtree. For the leaf node, there is additional information of a predicted class label. Starting from the root of a shapelet decision tree classifier, we calculate the distance from the testing object T to the shapelet in that node. If the distance is smaller than the split point, we recursively use the left subtree (lines 6 and 7) and otherwise use the right subtree (lines 8 and 9). This procedure continues until we reach the leaf node and return the predicted class label (lines 1 and 2).

Predict (shapelet decision tree classifier **C**, testing time series T)	
1	**If C** is the leaf node
2	**Return** label of **C**
3	**Else**
4	S ← shapelet on the root node of **C**
5	*split_point* ← split point on the root of **C**
6	**If** SubsequenceDistanceEarlyAbandon (T, S) < *split_point*
7	Predict (left substree of **C**, T)
8	**Else**
9	Predict (right substree of **C**, T)
10	**EndIf**
11	**EndIf**

Table 9. *Predicting the class label of a testing object.*

5 EXPERIMENTAL EVALUATION

We begin by discussing our experimental philosophy. We have designed and conducted all experiments such that they are easily reproducible. With this in mind, we have built a webpage[3] which contains all of the datasets and code used in this work, together with spreadsheets which contain the raw numbers displayed in all of the figures, and larger annotated figures showing the decision trees, etc. In addition, this webpage contains many additional experiments which we could not fit into this work; however, we note that this paper is completely self-contained.

Projectile point classification continues to be of use in archaeology. Projectile points can be divided into classes based on the location they are found, the group that created or used them, the dates they were in use, etc. In figure 12, we show some examples of the projectile points used in our experiments.

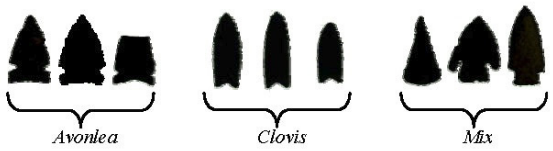

Figure 12. *Examples of the three classes of projectile points in our dataset. The testing dataset includes some broken points, and some line drawings.*

We converted the shapes of projectile points to a time series using the angle-based method.[4] We then randomly created a 36/175 training/test split. The results are shown in figure 13.

[1]Leo Breiman et al., *Classification and Regression Trees* (Belmont, CA: Wadsworth, 1984).

[2]Pierre Geurts, "Pattern Extraction for Time Series Classification," *Proceedings of the 5th Conference on Principles and Practice of Knowledge Discovery in Databases* (2005) 115–127.

[3]www.cs.ucr.edu/~lexiangy/shapelet.html.

[4]Eamonn Keogh et al., "LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006) 882–893.

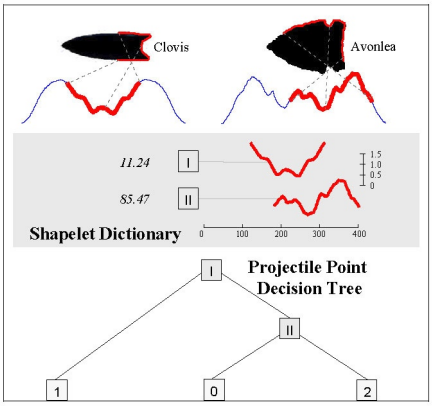

Figure 13. (top) *The dictionary of shapelets, together with the thresholds d_{th}.* (bottom) *The decision tree for the 3-class projectile points problem.*

The Clovis projectile points can be distinguished from the others by an un-notched hafting area near the bottom connected by a deep concave base. After distinguishing the Clovis projectile points, the Avonlea points are differentiated from the mixed class by a small notched hafting area connected by a shallow concave base.

The shapelet decision tree classifier achieves an accuracy of 80.0%, whereas the accuracy of *rotation invariant* one-nearest-neighbor classifier is 68.0%. Beyond the advantage of greater accuracy, the shapelet decision tree classifier produces the classification result 3×10^3 times faster than the *rotation invariant* one-nearest-neighbor classifier, and it is more robust in dealing with the pervasive broken projectile points in most collections.

6 Conclusions and Future Work

We have introduced a new primitive for time series and shape mining, *time series shapelets*. We have shown with extensive experiments that we can find the shapelets efficiently, and that they can provide accurate, interpretable and fast classification decisions for the discrimination of projectile points. Ongoing and future work includes extensions to the multivariate case and detailed case studies in the domains of anthropology and MOCAP analyses.

Appendix: A Note on Notation

The text of this paper features 'Big *O*' notation in the introduction section, such as "The classification time is just O(*ml*)." Big *O* notation is used in computer science to describe the computational complex of the algorithms. The notation explains the increaseing speed of the computational time of the algorithm as its input size increases. So O(*n*) means the computational time increases linearly with the input size, which means, if you double the input size, then the time you get the final answer is also doubled. O(n^3) indicates the computational time increases cubically with the input size; if you double the input size, then the time you finish computing is eight time longer. Figure 14 shows the how the running time increases on input size under different big *O* notations.

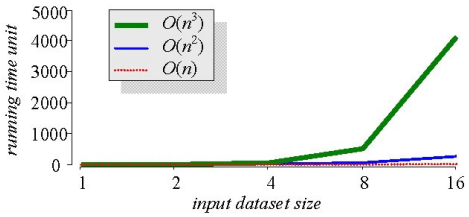

Figure 9. *Different running time increasing speed as the input size increases under different big O notation.*

Acknowledgements

This work was funded by NSF 0803410 and 0808770.

Bibliography

Black, Glenn A., and Paul Weer. "A Proposed Terminology for Shape Classifications of Artifacts," *American Antiquity* 1 (1936): 280–294.

Breiman, Leo, et al. *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

Buchanan, Briggs. "An Analysis of Folsom Projectile Point Resharpening Using Quantitative Comparisons of Form and Allometry," *Journal of Archaeological Science* 33 (2006): 185–199.

Buchanan, Briggs, and Mark Collard. "Investigating the peopling of North America through Cladistic Analyses of Early Paleoindian Projectile Points," *Journal of Anthropological Archaeology* 26 (2007): 366–393.

———. "Phenetics, Cladistics, and the Search for the Alaskan Ancestors of the Paleoindians: A Reassessment of Relationships among the Clovis, Nenana, and Denali Archaeological Complexes," *Journal of Archaeological Science* 35 (2008): 1683–1694.

Chiu, Bill, et al. "Probabilistic Discovery of Time Series Motifs," *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2003) 493–498.

Ding, Hui, et al. "Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures," *Proceedings of the 34th International Conference on Very Large Data Bases* (2008) 1542–1552.

Finkelstein, J. Joe. "A Suggested Projectile-Point Classification," *American Antiquity* 2 (1937): 197–203.

Geurts, Pierre. "Pattern Extraction for Time Series Classification," *Proceedings of the 5th Conference on Principles and Practice of Knowledge Discovery in Databases* (2001) 115–127.

Keogh, Eamonn, and Shruti Kasetty. "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2005): 102–111.

Keogh, Eamonn, et al. "LB_Keogh Support Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures," *Proceedings of the 32nd International Conference on Very Large Data Bases* (2006) 882–893.

O'Brien, Michael J., et al. "Cladistics Is Useful for Reconstructing Archaeological Phylogenies: Palaeoindian Points from the Southeastern United States," *Journal of Archaeological Science* 28 (2001): 1115–1136.

———. "Two Issues in Archaeological Phylogenetics: Taxon Construction and Outgroup Selection," *Journal of Theoretical Biology* 215 (2002): 133–150.

Salzberg, Steven L. "On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach." *Data Mining and Knowledge Discovery* 1 (1997): 317–328.

Shott, Michael J. "Stones and Shafts Redux: The Metric Discrimination of Chipped-Stone Dart and Arrow Points," *American Antiquity* 62 (1997): 86–101.

Thomas, David Hurst. "Arrowheads and Atlatl Darts: How the Stones Got the Shaft," *American Antiquity* 43 (1978): 461–472.

———. "How to Classify the Projectile Points from Monitor Valley, Nevada," *Journal of California and Great Basin Anthropology* 3 (1981): 7–43.

Wilcoxon, Frank. "Individual Comparisons by Ranking Methods," *Biometrics* 1 (1945): 80–83.

Xi, Xiaopeng, et al. "Fast Time Series Classification Using Numerosity Reduction," *Proceedings of the 23rd International Conference on Machine Learning* 148 (2006): 1033–1040.