The Great Time Series Classification Bake Off: A Review and Experimental Evaluation of Recently Proposed Algorithms

Anthony Bagnall, Aaron Bostrom, James Large and Jason Lines

University of East Anglia
Norwich, Norfolk
United Kingdom

ABSTRACT

In the last five years there have been a large number of new time series classification algorithms proposed in the literature. These algorithms have been evaluated on subsets of the 47 data sets in the University of California, Riverside time series classification archive. The archive has recently been expanded to 85 data sets, over half of which have been donated by researchers at the University of East Anglia. Aspects of previous evaluations have made comparisons between algorithms difficult. For example, several different programming languages have been used, experiments involved a single train/test split and some used normalised data whilst others did not. The relaunch of the archive provides a timely opportunity to thoroughly evaluate algorithms on a larger number of datasets. We have implemented 20 recently proposed algorithms in a common Java framework and compared them against two standard benchmark classifiers (and each other) by performing 100 resampling experiments on each of the 85 datasets. We use these results to test several hypotheses relating to whether the algorithms are significantly more accurate than the benchmarks and each other. Our results indicate that only 9 of these algorithms are significantly more accurate than both benchmarks and that one classifier, the Collective of Transformation Ensembles, is significantly more accurate than all of the others. All of our experiments and results are reproducible: we release all of our code, results and experimental details and we hope these experiments form the basis for more rigorous testing of new algorithms in the future.

1. INTRODUCTION

Time series classification (TSC) problems are differentiated from traditional classification problems because the attributes are ordered. Whether the ordering is by time or not is in fact irrelevant. The important characteristic is that there may be discriminatory features dependent on the ordering. The introduction of the UCR time series classification and clustering repository [21] saw a rapid growth in the number of publications proposing time series classification algorithms. Prior to the summer of 2015 over 3,000 people have downloaded the UCR archive and it has been referenced several hundred times. The repository has contributed to increasing the quality of evaluation of new TSC algorithms. Most experiments involve evaluation on over forty data sets, often with correct significance testing and most authors release source code. This degree of evaluation and reproducibility is generally better than most areas of machine learning and data mining research.

However, there are still some fundamental problems with published TSC research that we aim to address. Firstly, nearly all evaluations are performed on a single train/test split. This can lead to over interpreting of results. The majority of machine learning research involves repeated resamples of the data, and we think TSC researchers should follow suit. To illustrate why, consider the following anecdote. We were recently contacted by a researcher who queried our published results for one nearest neighbour (1-NN) dynamic time warping (DTW) on the UCR train test splits. When comparing our accuracy results to theirs, they noticed that in some instances they differed by as much as 6%. Over all the problems there was no significant difference, but clearly we were concerned, as it is a deterministic algorithm. On further investigation, we found out that our data were rounded to six decimal places, there data to eight. These differences on single splits were caused by small data set sizes and tiny numerical differences. When resampling, there were no significant differences on individual problems when using 6 or 8 decimal places.

Secondly, there are some anomalies and discrepancies in the UCR data that can bias results. Not all of the data are normalised (e.g. Coffee) and some have been normalised incorrectly (e.g. ECG200). This can make algorithms look better than they really are. For example, most authors cite an error of 17.9% for the Coffee data with 1-NN DTW, and most algorithms easily achieve lower error. However, 17.9% error is for DTW on the non-normalised data. If it is normalised, 1-NN DTW has 0% error, a somewhat harder benchmark to beat. ECG200 has been incorrectly formatted so that the sum of squares of the series can classify the data perfectly. If a classifier uses this feature it should be completely accurate. This will be a further source of bias.

Thirdly, the more frequently a fixed set of problems is used, the greater the danger of overfitting and detecting significant improvement that does not generalise to new problems. We should be constantly seeking new problems and enhancing the repository with new data. This is the only real route to detecting genuine improvement in classifier performance.

Finally, whilst the release of source code is admirable, the fact there is no common framework means it is often very hard to actually use other peoples code. We have reviewed algorithms written in C, C++, Java, Matlab, R and python. Often the code is "research grade", i.e. designed to achieve the task with little thought to reusability or comprehensibility. There is also the tendency to not provide code that performs model selection, which can lead to suspicions that parameters were selected to minimize test error, thus biasing the results.

To address these problems we have implemented 20 different TSC algorithms in Java, integrated with the WEKA toolkit [17]. We have applied the following guidelines for the inclusion of an algorithm. Firstly, the algorithm must have been recently published in a high impact conference or journal. Secondly, it must have been evaluated on some subset of the UCR data. Thirdly, source code must be available. Finally, it must be feasible/within our ability to implement the algorithm in Java. Often, variants of a classifier are described within the same publication. We have limited each paper to one algorithm and taken the version we consider most representative of the key idea behind the approach.

We have conducted experiments with these algorithms and standard WEKA classifiers on 100 resamples of the 85 data sets, each of which is normalised. In addition to resampling the data sets, we have also conducted extensive model selection for many of the classifiers. This is one of the largest ever experimental studies conducted in machine learning. We have performed millions of experiments distributed over thousands of nodes of a large high performance computing facility. Nevertheless, the goal of the study is tightly focussed and limited. This is meant to act as a springboard for further investigation into a wide range of TSC problems we do not address. Specifically, we assume all series in a problem are equal length, real valued and have no missing values. Classification is offline, and we assume the cases are independent (i.e. we do not perform streaming classification). All series are labelled and all problems involve learning the labels of univariate time series. We are interested in testing hypotheses about the average accuracy of classifiers over a large set of problems. Algorithm efficiency and scalability are of secondary interest at this point. Detecting whether a classifier is on average more accurate than another is only part of the story. Ideally, we would like to know *a priori* which classifier is better for a class of problem or even be able to detect which is best for a specific data set. However, this is beyond the scope of this paper.

Our findings are surprising, and a little embarrassing, for two reasons. Firstly, many of the algorithms are in fact no better than our two benchmark classifiers, 1-NN DTW and Rotation Forest. Secondly, of those X significantly better than both benchmarks, by far the best classifier is COTE [2], an algorithm we proposed. It is on average over 8% more accurate than both benchmarks. Whilst gratifying for us, we fear that this outcome may cause some to question the validity of the study. We have made every effort to faithfully reproduce all algorithms. We have tried to reproduce published results, with varying degrees of success (as described below), and have consulted authors on the implementation where possible. Our results are reproducible, and we welcome all input on improving the code base. We must stress that COTE is by no means the final solution. All of the algorithms we describe may have utility in specific domains, and many are orders of magnitudes faster than COTE. Nevertheless, we believe that it is the responsibility of the designers of an algorithm to demonstrate its worth. We think our benchmarking results will help facilitate an improved understanding of utility of new algorithms under alternative scenarios. The code is freely accessible from a repository [1] and detailed results and data sets are available from a dedicated website [3].

The rest of this paper is structured as follows. In Section 2 we review the algorithms we have implemented. In Section 3 we describe the data, code structure and experimental design. In Section 4 we present and analyse the results, and in Section 5 we summarise our findings and discuss the future direction.

2. CLASSIFICATION ALGORITHMS

We denote a vector in bold and a matrix in capital bold. A case/instance is a pair $\{\mathbf{x}, y\}$ with m observations $x_1, \ldots, x_m$ (the time series) and discrete class variable y with c possible values. A list of n cases with associated class labels is $\mathbf{T} =< \mathbf{X}, \mathbf{y} >=< (\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n) >$. A classifier is a function or mapping from the space of possible inputs to a probability distribution over the c class variable values.

The large majority of time series research in the field of data mining has concentrated on alternative distance measures that can be used for clustering, query and classification. For TSC, these distance measures are almost exclusively evaluated with a one nearest neighbour (1-NN) classifier. The standard benchmark distance measures are Euclidean distance (ED) and dynamic time warping (DTW). Alternative techniques taken from other fields include edit distance with real penalty (ERP) and longest common subsequence (LCSS). Three more recently proposed time domain distance measures are described in Section 2.1. DTW is by far the most popular benchmark. It is common to put a restriction on the amount of warping allowed. This restriction is equivalent to putting a maximum allowable distance between any pairs of indexes in a proposed path. It has been shown that setting the proportion of warping allowed, r, through cross validation to maximize training accuracy, as proposed in [28], significantly increases accuracy [24]. We set r through cross validation in all out experiments.

2.1 Time Domain Distance Based Classifiers

In 2008, Ding *et al.* [12] evaluated 8 different distance measures on 38 data sets and found none significantly better than DTW. Since then, three more elastic measures have been proposed.

Weighted DTW (WDTW) [19]. Jeong *et al.* describe WDTW, which adds a multiplicative weight penalty based on the warping distance between points in the warping path. It favours reduced warping, and is a smooth alternative to the cutoff point approach of using a warping window. A logistic weight function is used, with a parameter g that controls the penalty level for large warpings.

Time Warp Edit (TWE) [25]. Marteau proposes TWE distance, an elastic distance metric that includes characteristics from both LCSS and DTW. It allows warping in the time axis and combines the edit distance with Lp-norms. The warping is controlled by a *stiffness* parameter, ν. Stiffness enforces a multiplicative penalty on the distance between matched points in a manner similar to WDTW. A penalty value λ is applied when sequences do not match.

Move-Split-Merge (MSM) [32]. Stefan *et al.* present MSM distance, a metric that is conceptually similar to other edit distance-based approaches, where similarity is calculated by using a set of operations to transform a given series into a target series. Move is synonymous with a substitute operation, where one value is replaced by another. Split and merge differ from other approaches, as they attempt to add context to insertions and deletions. The split operation inserts an identical copy of a value immediately after itself, and the merge operation is used to delete a value if it directly follows an identical value.

We have implemented WDTW, TWE, MSM and other commonly used time domain distance measures, which are available in the package weka.core.elastic_distance_measures. We have generated results that are not significantly different to those published when using these distances with 1-NN.

2.2 Differential Distance Based Classifiers

There are a group of algorithms that are based on the first order differences of the series, $a'_i = a_i - a_{i+1}$. Various methods that have used a form of differences have been described [19], but the most successful approaches combine distance in the time domain and the difference domain.

Complexity Invariant distance (CID) [4]. Batista *et al.* describe a means of weighting a distance measure to compensate for differences in the complexity in the two series being compared. Any measure of complexity can be used, but Batista *et al.* recommend the simple expedient of using the sum of squares of the first differences. If there is a large difference in complexity between the two series then the distance increases.

Derivative DTW (DD$_{DTW}$) [14]. Górecki and Łuczak describe an approach for using a weighted combination of raw series and first-order differences for NN classification with either the Euclidean distance or full-window DTW. They find the DTW distance between two series and the two differenced series. These two distances are then combined using a weighting parameter α. Parameter α is found during training through a leave-one-out cross-validation on the training data. This search is relatively efficient as different parameter values can be assessed using pre-computed distances. An optimisation to reduce the search space of possible parameter values is proposed in [14]. However, we could not recreate their results using this optimisation. We found that if we searched through all values of α in the range of $[0, 1]$ in increments of 0.01, we were able to recreate the results exactly. Testing is then performed with a 1-NN classifier using the combined distance function.

Derivative Transform Distance (DTD$_C$) [15]. Górecki and Łuczak proposed an extension of DD$_{DTW}$ that uses DTW in conjunction with derivatives and transforms. They propose and evaluate combining DD$_{DTW}$ with distances on data transformed with the sin, cosine and Hilbert transform. We implement the cosine version. Distances in time domain, difference and cosine are combined with two weighting parameters, α and β.

DD$_{DTW}$ was evaluated on single train test splits of 20 UCR datasets, CID$_{DTW}$ on 43 datasets and DTD$_C$ on 47. We can recreate results that are not significantly different to those published for all three algorithms. All papers claim superiority to DTW. The small sample size for DD$_{DTW}$ makes this claim debatable, but the published results for CID$_{DTW}$ and DTD$_C$ are both significantly better than DTW. On published results, DTD$_C$ is significantly more accurate than CID$_{DTW}$ and CID$_{DTW}$ is significantly better than DD$_{DTW}$.

2.3 Dictionary Based Classifiers

Dictionary based approaches approximate and reduce the dimensionality of series by transforming them into representative words, then basing similarity on comparing the distribution of words. The core process of dictionary approaches involves forming words by passing a sliding window, length w, over each series, approximating each window to produce l values, and then discretising these values by assigning each a symbol from an alphabet of size α. The occurrence of words is used to form histograms that are then representative of series or classes. The parameters are set through cross validation.

Bag of Patterns (BOP) [23]. BOP is a dictionary classifier built on the Symbolic Aggregate Approximation (SAX) method for converting series to strings [22]. SAX reduces the dimension of a series through Piecewise Aggregate Approximation (PAA) [8], then discretises the series into bins formed from equal probability areas of the Normal distribution. BOP works by applying SAX to each window to form a word. If consecutive windows produce identical words, then it is deemed a trivial match and only the first of that run is recorded. The distribution of words over a series forms a count histogram. New cases are classified with a 1-NN Euclidean distance classifier applied to the histograms.

Symbolic Aggregate Approximation - Vector Space Model (SAXVSM) [31]. SAXVSM combines the SAX representation used in BOP with the vector space model commonly used in Information Retrieval. The key differences between BOP and SAXVSM is that SAXVSM forms word distributions over classes rather than series and weights these by the term frequency/inverse document frequency ($tf \cdot idf$). For SAXVSM, term frequency refers to the number of times a word appears in a class and document frequency means the number of classes a word appears in. Predictions are made using a 1-NN classification based on the word frequency distribution of the new case and the $tf \cdot idf$ vectors of each class. Cosine similarity measure is used.

Bag of SFA Symbols (BOSS) [30]. BOSS also uses windows to form words over series, but it has several major differences to BOP and SAXVSM. Primary amongst these is that BOSS uses a truncated Discrete Fourier Transform (DFT) instead of a PAA on each window; the truncated series is discretised through a technique called Multiple Coefficient Binning (MCB), rather than using fixed intervals; and BOSS ensembles alternative versions of the base algorithm. MCB finds the disretising break points as a preprocessing step by estimating the distribution of the Fourier coefficients. This is performed by segmenting the series, performing a DFT, then finding breakpoints for each coefficient so that each bin contains the same number of elements. BOSS then involves similar stages to BOP; it windows each series to form word distribution through the application of DFT and discretisation by MCB. A bespoke distance function is used for nearest neighbour classification. This non symmetrical function only includes distances between frequencies of words that actually occur within the first histogram passed as an argument. BOSS also includes a parameter that determines whether the subseries are normalised or not.

DTW Features ($\mathbf{DTW}_F$) [20]. Kate proposes a feature generation scheme that combines DTW distances to training cases and SAX histograms. A training set with n cases is transformed into a set with n features, where feature x_{ij} is the full window DTW distance between case i and case j. A further n features are then created. These are the optimal window DTW distance between cases. Finally, SAX word frequency histograms are created for each instance using the BOP algorithm. These a^l features are concatenated with the $2n$ full and optimal window DTW features. The new data set is trained with a support vector machine with a polynomial kernel with order either 1, 2 or 3, set through cross validation. DTW window size and SAX parameters are also set independently through cross validation with a 1-NN classifier.

BOP and SAXVSM were evaluated on the 20 and 19 UCR problems respectively. All algorithms used the standard single train/test split. BOSS presents results on an extended set of 58 data sets from a range of sources, DTW_F uses 47 UCR data. On the 19 data sets they all have in common, BOP is significantly worse than BOSS and SAXVSM. There is no significant difference between DTWF, BOSS and SAXVSM. Furthermore, there is no significant difference between BOSS and DTW_F on the 44 datasets they have in common. Our BOP and DTW_F results are not significantly different to the published ones. We were unable to reproduce as accurate results as published for SAXVSM and BOSS. On examination of the implementation for SAXVSM provided online and by correspondence with the author, it appears the parameters for the published results were obtained through optimisation on the test data. Our results for BOSS are on average approximately 1% worse than those published, a significant difference. Correspondence with the author and examination of the code leads us to believe this is because of a versioning problem that meant that the normalisation parameter was set to minimize test data error rather than train error. This would introduce significant bias.

2.4 Shapelet Based Classifiers

Shapelets are time series subseries that are discriminatory of class membership. They allow for the detection of phase-independent localised similarity between series within the same class. The original shapelets algorithm by Ye and Keogh [33] uses a shapelet as the splitting criterion for a decision tree. There have been three recent advances in using shapelets.

Fast Shapelets (FS) [27]. Rakthanmanon and Keogh propose an extension of the decision tree shapelet approach [33, 26] that speeds up shapelet discovery. Instead of a full enumerative search at each node, the fast shapelets algorithm discretises and approximates the shapelets. Specifically, for each possible shapelet length, a dictionary of SAX words is first formed. The dimensionality of the SAX dictionary is reduced through masking randomly selected letters (random projection). Multiple random projections are performed, and a frequency count histogram is built for each class. A score for each SAX word can be calculated based on how well these frequency tables discriminate between classes. The k-best SAX words are selected then mapped back to the original shapelets, which are assessed using information gain in a way identical to that used in [33].

Shapelet Transform (ST) [18, 7]. Hills *et al.* describe the shapelet transformation that separates the shapelet discovery from the classifier by finding the top k shapelets on a single run (in contrast to the decision tree, which searches for the best shapelet at each node). The shapelets are used to transform the data, where each attribute in the new dataset represents the distance of a series to one of the shapelets. We use the most recent version of this transform [7] that balances the number of shapelets per class and evaluates each shapelet on how well it discriminates just one class. Following [2, 7] we construct a classifier from the shapelet transformed dataset using a weighted ensemble of standard classifiers. We include k Nearest Neighbour (where k is set through cross validation), Naive Bayes, C4.5 decision tree, Support Vector Machines with linear and quadratic basis function kernels, Random Forest (with 500 trees), Rotation Forest (with 50 trees) and a Bayesian network. Each classifier is assigned a weight based on the cross validation training accuracy, and new data (after transformation) are classified with a weighted vote. With the exception of k-NN, we use default parameters for these classifiers.

Learned Shapelets (LS) [16]. Grabocka *et al.* describe a shapelet discovery algorithm that adopts a heuristic gradient descent shapelet search procedure rather than enumeration. LS finds k shapelets that, unlike the alternatives, are not restricted to being subseries in the training data. The k shapelets are initialised through a k-means clustering of candidates from the training data. The objective function for the optimisation process is a logistic loss function (with regularization term) L based on a logistic regression model for each class. The algorithm jointly learns the weights for the regression $\mathbf{W}$, and the shapelets $\mathbf{S}$ in a two stage iterative process to produce a final logistic regression model. A check is performed at certain intervals as to whether divergence has occurred. This is defined as a train set error of 1 or infinite loss. The check is performed when half the number of allowed iterations is complete. This criteria meant that for some problems, LS never terminated during model selection. Hence we limited the the algorithm to a maximum of five restarts.

FS, LS and ST were evaluated on 33, 45 and 75 data sets respectively. We can reproduce results that are not significantly different to FS and ST. The published results for FS are significantly worse than those for LS and ST. There is no significant difference between the LS and ST published results. We can reproduce the output of the code released for LS but are unable to reproduce the actual published results. The author of LS believes the difference is caused by the fact we have not included the adaptive learning rate adjustment implemented through Adagrad. We are working with him to include this enhancement.

2.5 Interval Based Classifiers

A family of algorithms derive features from intervals of each series. For a series of length m, there are $m(m-1)/2$ possible contiguous intervals. The two key decisions about using this approach are, firstly, how to deal with the huge increase in the dimension of the feature space and secondly, what to actually do with each interval. Rodriguez *et al.* [29] were the first to adopt this approach and address the first issue by using only intervals of lengths equal to powers of two and the second by calculating binary features over each intervals based on threshold rules on the interval mean and standard deviation. A support vector machine is then trained on this transformed feature set. This algorithm was a precursor to three recently proposed interval based classifiers that we have implemented.

Time Series Forest (TSF) [11]. Deng *et al.* [11] overcome the problem of the huge interval feature space by employing a random forest approach, using summary statistics (mean, standard deviation and slope) of each interval as features. Training a single tree involves selecting $\sqrt{m}$ random intervals, generating the mean, standard deviation and slope of the random intervals then creating and training a tree on the resulting $3\sqrt{m}$ features. Classification is by a majority vote of all the trees in the ensemble. A classification tree that has two bespoke characteristics is defined in [11]. Firstly, rather than evaluate all possible split points to find the best information gain, a fixed number of evaluation points is pre-defined. We assume this is an expedient to make the classifier faster, as it removes the need to sort the cases by each attribute value. Secondly, a refined splitting criteria to choose between features with equal information gain is introduced. This is defined as the distance between the splitting margin and the closest case. The intuition behind the idea is that if two splits have equal entropy gain, then the split that is furthest from the nearest case should be preferred. This measure would have no value if all possible intervals were evaluated because by definition the split points are taken as equi-distant between cases. We experimented with including these two features, but found the effect on accuracy was, if anything, negative. We found the computational overhead of evaluating all split points acceptable, hence we had no need to include the margin based tie breaker. We used the built in Weka RandomTree classifier (which is the basis for the Weka RandomForest classifier) with default parameters. This means there is no limit to the depth of the tree nor a minimum number of cases per leaf node.

Time Series Bag of Features (TSBF) [6]. TSBF is an extension of TSF that has multiple stages. There are some aspects of TSBF we do not have space to fully explain. However, the description captures the essential features. The first stage involves generating a subseries classification problem. This involves selecting random subseries, then finding summary statistics (mean, standard deviation and slope) over intervals of each subseries. If w subseries are are selected the internal classification problem will have $w \cdot n$ cases. A random forest classifier is trained on this new problem. The out of bag error of this classifier is used to form a probability distribution over the c possible class values for each $w \cdot n$ case. These probabilities are discretised into equal width bins. A bag of features for each original instance is formed from these discretised words, giving a count histogram for each case. Finally, a random forest classifier is built on the bag of features representation. New cases are classified by following the same stages of transformation and internal classification. The number of subseries and the number of intervals are determined by a parameter, z. Training involves searching possible values of z for the one that minimizes the out of bag error for the final classifier. Other parameters are fixed for all experiments.

Learned Pattern Similarity (LPS) [5]. LPS was developed by the same research group as TSF and TSBF at Arizona State University. It is also based on intervals, but the main difference is that subseries become attributes rather than cases. Like TSBF, building the final model involves first building an internal predictive model. However, LPS creates an internal regression model rather than a classification model. The internal model is designed to detect correlations between subseries, and in this sense is an approximation of an autocorrelation function. LPS selects random subseries. For each location, the subseries in the original data are concatenated to form a new attribute. The process is repeated and the first order difference of the subseries is used. The internal model selects a random attribute as the response variable then constructs a regression tree. A collection of these regression trees are processed to form a new set of instances based on the counts of the number of subseries at each leaf node of each tree. There are two versions of LPS available, both of which aim to avoid the problem of generating all possible subseries. The R and C version creates the randomly selected attribute at Stage 1 on the fly at each level of the tree. This avoids the need to generate all possible subseries, but requires a bespoke tree. The second implementation (in Matlab) fixes the number of subseries to randomly select for each tree. Experiments suggest there is little difference in accuracy between the two approaches. We adopt the latter algorithm because it allows us to use the Weka RandomRegressionTree algorithm, thus simplifying the code and reducing the likelihood of bugs.

TSF and TSBF were evaluated on the original 47 UCR problems, LPS on an extended set of 75 data sets first used in [24] using the standard single train/test splits. The ranks of the published results show that although TSBF has the highest average rank, there is no significant difference between the classifiers at the 5% level. Pairwise comparisons yield no significant difference between the three.

We can reproduce results that are not significantly different to those published for TSF and LPS. Our TSBF results are significantly worse than those published. The mean difference is just over 1%. TSBF is a complex algorithm, and it is possible there is a mistake in our implementation, but our best debugging efforts were not able to find one. It may be caused by a difference in the random forest implementations of R and Weka or by an alternative model selection method.

2.6 Ensemble Classifiers

Ensembles have proved popular in recent TSC research and are highly competitive with general classification problems. TSF, TSBF and BOSS are ensembles based on the same core classifier. Other approaches, such as the *ST* ensemble described in Section 2.4, use different classifier components. Two other recently proposed heterogenous TSC ensembles are as follows.

Elastic Ensemble (EE) [24]. The EE is a combination of nearest neighbour (NN) classifiers that use elastic distance measures. Lines and Bagnall [24] show that none of the individual components of EE significantly outperforms DTW. However, we demonstrate that by combining the predictions of 1-NN classifiers built with these distance measures and using a voting scheme that weights according to cross-validation training set accuracy, we can significantly outperform DTW. The 11 classifiers in EE are 1-NN with Euclidean distance, DTW (full and CV window), derivative DTW (full and CV window), WDTW and derivative weighted DTW [19], LCSS, Edit Distance with Real Penalty, TWE distance [25], and MSM distance metric [32].

Collective of Transformation Ensembles (COTE) [2]. Bagnall *et al.* propose the meta ensemble COTE, a combination of classifiers in different transformation domains. The components of EE and ST are pooled with classifiers built on a version of autocorrelation transform (ACF) and power spectrum (PS) transform. EE uses the 11 classifiers described above. ACF and PS employ the same 8 classifiers used by with the shapelet transform (see Section 2.4). We use the classifier called flat-COTE in [2]. This involves pooling all 35 classifiers into a single ensemble with votes weighted by train set cross validation accuracy.

We have grouped the algorithms for clarity, but the classifications are overlapping. For example, TSBF is an interval based and ensemble based approach and LPS is based on auto-correlation. Table 1 gives the break down of algorithm verses approach.

There are many other approaches that have been proposed that we have not included due to time constraints and failure to meet our inclusion criteria. Two worthy of mention are Silva *et al.*'s Recurrence Plot Compression Distance (RPCD) [9] and Fulcher and Jones's feature-based linear classifier (FBL) [13]. RPCD involves trsansforming each series into a 2 dimensional recurrence plot then measuring similarity based on the size of the MPEG1 encoding of the concatenation of the resulting images. We were unable to find a working Java based MPEG1 encoder, and the technique seems not to work with the MPEG4 encoders we tried. FBL involves generating a huge number of possible features which are filtered with a forward selection mechanism for a linear classifier. The technique utilises built in matlab functions to generate thousands of features. Unfortunately these functions are not readily available in Java, and we considered it infeasible to attempt such as colossal task.

Table 1: A summary of algorithms and the component approaches underlying them. Approaches are nearest neighbour classification (NN), time domain distance function (time), derivative based distance function (deri), shapelet based (shpt), interval based (int), dictionary based (dict), autocorrelation/spectral based (auto) and ensemble (ens)

	NN	time	deri	shpt	int	dict	auto	ens
WDTW	x	x						
TWE	x	x						
MSM	x	x						
CID	x	x	x					
DD$_{DTW}$	x	x	x					
DTD$_C$	x	x	x					
ST				x				x
LS				x				
FS				x				
TSF					x			x
TSBF					x			x
LPS	x		x		x		x	
BOP	x					x		
SAXVSM	x					x		
BOSS	x					x	x	x
DTW$_F$	x	x				x		x
EE	x	x	x					x
COTE	x	x	x	x			x	x

3. DATA AND EXPERIMENTAL DESIGN

The 85 datasets are described in detail on the website [3]. The collection is varied in terms of data characteristics: the length of the series ranges from 24 (ItalyPowerDemand) to 2709 (HandOutlines); train set sizes vary from 16 to 8926; and the number of classes is between 2 and 60. The data are from a wide range of domains, with an over representation of image outline classification problems (29 problems). Other categories are sensor readings (16), motion capture (14), food spectrographs (7), ECG measurements (7), electric device profiles (6) and simulated data (6). Four of the spectrographs data sets have not been used for TSC before: Ham; Meat; Strawberry; and Wine. These were all created by the Institute of Food Research, part of the Norwich Research Park, as were the three spectra data already in the UCR (Beef, Coffee and OliveOil).

We run the same 100 resample folds on each problem for every classifier. The first fold is always the original train test split. The other resamples are stratified to retain class distribution in the original data sets. These resample datasets can be exactly reproduced. Each classifier must be built and evaluated 8,500 times. Model selection is repeated on every training set fold. We searched the parameter value spaces defined in the relevant publication as closely as possible. The parameter values we search are available in the longer version of this paper [3]. We allow each classifier a maximum 100 parameter values, each of which we assess through a cross validation on the training data. The number of cross validation folds is dependent on the algorithm. This is because the overhead of the internal cross validation differs. For the distance based measures it is as fast to do a leave-one-out cross validation as any other. For others we need a new model for each set of parameter values. This means we need to construct 850,000 models for each classifier. When we include repetitions caused by bugs, we estimate we have conducted over 30 million distinct experiments over six months.

The datasets vary greatly in size. We have had to sub-sample the eight largest data sets for the model selection stages, in particular for the slower algorithms such as ST, LPS and BOSS. Full details of the sampling performed are in the code documentation. We follow the basic methodology described in [10] when testing for significant difference between two or more classifiers. Our main focus of interest is relative performance over multiple data sets. Hence, we average accuracies over all 100 resamples, then compare classifiers by ranks using the Friedman test and a *post-hoc* pairwise Nemenyi test to discover where the differences lie.

4. RESULTS

Due to space constraints, we present an analysis of our results rather than tabulate the full data. All of our results and spreadsheets to derive the graphs are available from [3].

4.1 Benchmark Classifiers

We believe that newly proposed algorithms should add some value in terms of accuracy or efficiency over sensible standard approaches which are generally much simpler and better understood. The most obvious starting point for any classification problem is to use a standard classifier that treats each series a vector. Some characteristics that make TSC problems hard include having few cases and long series (large number of attributes), many of which are redundant or correlated. These are problems that are well studied in machine learning and classifiers have been designed to compensate for them. TSC characteristics that will confound traditional classifiers include discriminatory features in the autocorrelation function, phase independence within a class and imbedded discriminatory subseries. However, not all problems will have this characteristic, and benchmarking against standard classifiers may give insights into the problem characteristics. We have conducted resample experiments with 11 possible benchmarks: logistic Regression (logistic); C4.5 (C45); naive Bayes (NB); Bayes net (BN); support vector machine with linear (SVML) and quadratic kernel (SVMQ); multilayer perceptron (MLP); 1-NN with Euclidean distance (ED) and Dynamic time warping (DTW); random forest (with 500 trees) (RandF); and rotation forest (with 50 trees) (RotF). The average ranks are shown in Figure 1. RotF, RandF and DTW form a clique of classifiers better than the others. Based on these results, we select RotF and DTW (with window size set through cross validation) as our two benchmarks classifiers.

4.2 Comparison Against Benchmark Classifiers

Table 2 shows the summary of the pairwise results of the 19 classifiers against DTW and RotF. Nine classifiers are significantly better than both benchmarks: COTE; ST; BOSS; EE; DTW$_F$; TSF; TSBF; LPS; and MSM. BOP, SAXVSM and FS are all significantly worse than both the benchmarks. This reflects the published FS results, but is worse than expected for BOP and SAXVSM.

4.3 Comparison of All TSC Algorithms

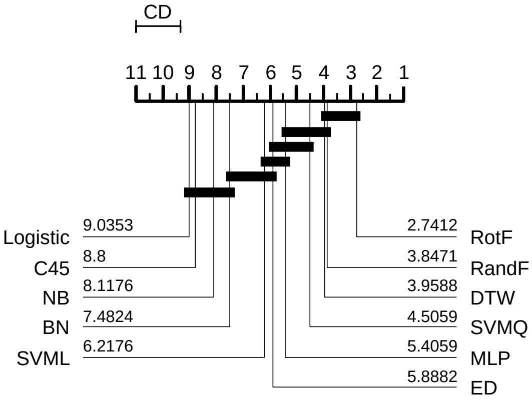

Figure 1: Critical difference diagram for 11 potential benchmark classifiers.

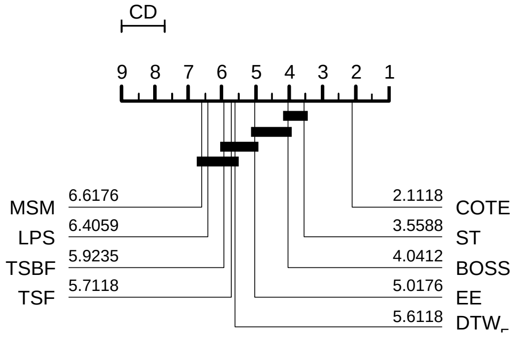

Figure 2: Critical difference diagram for the 9 classifiers significantly better than both benchmark classifiers.

Figure 2 shows the critical difference for the nine classifiers that are significantly better than both benchmarks. The most obivous conclusion from this graph is that COTE is significantly better than the others. EE and ST are components of COTE, hence this result demonstrates the benefits of combining classifiers on alternative feature spaces. The second distinguishing feature is the good performance of BOSS, and to a lesser degree, DTW$_F$. We discuss these results in detail below.

4.4 Results by Algorithm Type

Time Domain Distance Based Classifiers. Of the three distance based approaches we evaluated (TWE, WDTW and MSM), MSM is the highest rank (9^{th}) and is the only one significantly better than both benchmarks. WDTW (ranked 14^{th}) is better than DTW but not RotF. This conclusion contradicts the results in [24] which found no difference between all the elastic algorithms and DTW. This demonstrates that whilst there is a significant improvement, the benefit is small. MSM is under 2% on average better than DTW and RotF. The average of average differences in accuracy between WDTW and DTW is only 0.2%. The fact we are resampling has allowed us to detect such as small improvement. We made no attempts to speed up the distance measures, and of all the measures used in [24], MSM and TWE were by

Table 2: A summary of algorithm performance grouped based on significant difference to DTW and RotF. The column prop gives the proportion of problems where the classifier has a significantly higher mean accuracy over 100 resamples. The column mean gives the mean difference in mean accuracy over all 85 problems.

Classifier	Prop better	Mean difference	Classifier	Prop better	Mean difference
Significantly better than DTW			Significantly better than RotF		
COTE	96.47%	8.12%	COTE	84.71%	8.14%
EE	95.29%	3.51%	ST	75.29%	6.15%
BOSS	82.35%	5.76%	BOSS	63.53%	5.78%
ST	80.00%	6.13%	TSF	63.53%	1.93%
DTW$_F$	75.29%	2.87%	LPS	60.00%	1.86%
TSF	68.24%	1.91%	EE	58.82%	3.54%
TSBF	65.88%	2.19%	DTW$_F$	58.82%	2.89%
MSM	62.35%	1.89%	MSM	57.65%	1.91%
LPS	61.18%	1.83%	TSBF	56.47%	2.22%
WDTW	60.00%	0.20%	Not significantly different to RotF		
DTD$_C$	52.94%	0.79%	CID$_{DTW}$	48.24%	0.56%
CID$_{DTW}$	50.59%	0.54%	DTD$_C$	47.06%	0.82%
Not significantly different to DTW			DD$_{DTW}$	45.88%	0.44%
DD$_{DTW}$	56.47%	0.42%	TWE	45.88%	0.40%
RotF	56.47%	-0.02%	WDTW	44.71%	0.22%
TWE	49.41%	0.37%	LS	44.71%	-2.97%
Significantly worse than DTW			DTW	43.53%	0.02%
LS	47.06%	-2.99%	Significantly worse than RotF		
SAXVSM	41.18%	-3.29%	BOP	34.12%	-3.03%
BOP	37.65%	-3.05%	SAXVSM	31.76%	-3.26%
FS	30.59%	-7.40%	FS	22.35%	-7.38%

far the slowest. These results indicate it may be worthwhile examining speed ups for MSM.

Difference Based Classifiers. In line with published results, two of the difference based classifiers, CID$_{DTW}$ and DTD$_C$ are significantly better than DTW, but the mean improvement is very small (under 1%). None of the three approaches are significantly different to RotF. We believe this highlights an over-reliance on DTW as a benchmark. In line with the original description we set warping window for CID$_{DTW}$ as the optimal for DTW. Setting the window to optimise the CID$_{DTW}$ distance instead might well improve performance.

Dictionary Based Classifiers. The results for window based dictionary classifiers are confusing. SAXVSM and BOP are significantly worse than the benchmarks and ranked 18^{th} and 19^{th} overall respectively. This would seem to suggest there is little merit in dictionary transformations for TSC. However, the BOSS ensemble is one the most accurate classifiers we tested. It is significantly better than both benchmarks and is ranked third overall. The main differences between BOP and BOSS are that BOSS uses a Fourier transformation rather than PAA and employs a data driven discretisation rather than fixed boundaries. This indicates that there may be further scope for window based spectral classifiers. The use of an ensemble also significantly improves accuracy. It would be interesting to determine which difference contributes most to the improved performance of the BOSS ensemble. DTW$_F$ also did well (ranked 5^{th}). Including SAX features significantly improves DTW$_F$, so our conjecture is that the DTW features are compensating for the datasets that BOP does poorly on, whilst gaining from those it does well at. This would support the argument for combining features from different representations.

Shapelet Based Classifiers. FS is the least accurate classifier we tested and is significantly worse than the benchmarks. LS is not significantly better than either benchmark and in fact it is significantly worse than DTW. Our FS algorithm reproduces published results and we believe is faithful to the original. The LS results may be improved with the Adagrad enhancement that will be included. Conversely, the ST has exceeded our expectations. It is significantly better than both benchmarks and is the second most accurate classifier overall, significantly better than six of the other eight classifiers that beat both benchmarks. The changes proposed in [7] have not only made it much faster, but have also increased accuracy. Primary amongst these changes is balancing the number of shapelets per class and using a one-vs-many shapelet quality measure. However, ST is the slowest of all the algorithms we assessed and there is scope to increase the speed without compromising accuracy.

Interval Based Classifiers. The interval based approaches, TSF, TSBF and LPS, are all significantly better than both the benchmarks. This gives clear support to the idea of interval based approaches. There is no significant difference between them. Hence, based on this evidence, we conclude there is definite value in interval based algorithms and would favour TSF for its simplicity.

Ensemble Classifiers. The top seven classifiers are all ensembles. This is strong evidence to support the view that ensembling is one of the simplest ways of improving a classifier. It seems highly likely the other classifiers would benefit

Table 3: Best performing algorithms split by problem type. Each entry is the percentage of problems of that type a member of a class of algorithm is most accurate for.

Problem	COTE	Dictionary	Difference	Elastic	Interval	Shapelet	Vector	Counts
Image Outline	24.14%	13.79%	6.90%	17.24%	0.00%	17.24%	20.69%	29
Sensor Readings	38.89%	0.00%	5.56%	11.11%	5.56%	22.22%	16.67%	18
Motion Capture	35.71%	21.43%	7.14%	7.14%	14.29%	14.29%	0.00%	14
Spectrographs	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	100.00%	7
Electric Devices	0.00%	16.67%	0.00%	0.00%	16.67%	66.67%	0.00%	6
ECG measurements	33.33%	16.67%	0.00%	0.00%	0.00%	50.00%	0.00%	6
Simulated	40.00%	20.00%	0.00%	20.00%	0.00%	20.00%	0.00%	5
Overall	27.06%	11.76%	4.71%	10.59%	4.71%	22.35%	18.82%	
Counts	23	10	4	9	4	19	16	85

from a similar approach. One of the key ensemble design decisions is promoting diversity without compromising accuracy. TSF, TSBF and LPS do this through the standard approach of sampling the attribute space. BOSS ensembles identical classifiers with different parameter settings. ST and EE engender diversity though classifier heterogeneity. Employing different base classifiers in an ensemble is relatively unusual, and these results would suggest that it might be employed more often. COTE is significantly better than all other classifiers. It promotes diversity through employing different transformations/data representations in addition to using a range of base classifiers. Its simplicity is its strength. These experiments suggest COTE may be even more accurate if it were to assimilate BOSS and an interval based approach.

4.5 Results by Problem Type

Table 3 shows the performance of algorithms against problem type. The data is meant to give an indication as to which family of approaches may be best for each problem type. The sample sizes are small, so we must be careful drawing too many conclusions. However, this table does indicate how evaluation can give insights into problem domains. So, for example, Shapelets are best on 4 out of 6 of the ElectricDevice problems and 3 out of 6 ECG datasets, but only 26% of problems overall. This makes sense in terms of the applications, because the profile of electricity usage and ECG irregularity will be a subseries of the whole and largely phase independent. The vector classifiers do best on all of the Spectrograph data sets, indicating there is little discriminatory information in the location of measurements. COTE is the best algorithm on nearly 40% of the sensor problems. This suggests that there are a range of features that help classify these problems and no one representation is likely to be sufficient.

5. CONCLUSIONS

The primary goal of this series of benchmark experiments is to promote reproducible research and provide a common framework for future work in this area. We view data mining as a practical area of research, and our central motivation is to find techniques that work. Received wisdom is that DTW is hard to beat. Our results confirm this to a degree (7 out of 19 algorithms fail to do so), but recent advances show it not impossible. Our results indicate that COTE is, on average, clearly superior to other published techniques. It is on average 8% more accurate than DTW and RotF and 3% more accurate than the closest competitor. However, COTE is a starting point rather than a final solution. Firstly, the no free lunch theorem leads us to believe that no classifier will dominate all others. The research issues of most interest are what types of algorithm work best on what types of problem and can we tell *a priori* which algorithm will be best for a specific problem. Secondly, COTE is hugely computationally intensive. It is trivial to parallelise, but its run time complexity is bounded by the Shapelet Transform, which is $O(n^2m^4)$ and the parameter searches for the elastic distance measures, some of which are $O(n^3)$. An algorithm that is faster than COTE but not significantly less accurate would be a genuine advance in the field. Thirdly, COTE gives little insight into the problem domain. Finally, we are only looking at a very restricted type of problem. We have not considered multi-dimensional, streaming, windowed, long series or semi-supervised TSC, to name but a few variants. Each of these subproblems would benefit from a comprehensive experimental analysis of recently proposed techniques. We are constantly looking for new areas of application and we will include any new data sets that are donated in an ongoing evaluation. We will happily evaluate anyone else's algorithm if it is implemented as a WEKA classifier (with all model selection performed in the method buildClassifier) and if it is computationally feasible. If we are given permission we will release any results we can verify through the associated website. For those looking to build a predictive model for a new problem we would recommend starting with DTW, RandF and RotF as a basic sanity check and benchmark. We have made little effort to perform model selection for the forest approaches because it is generally accepted they are robust to parameter settings, but some consideration of forest size and tree parameters may yield improvements. However, our conclusion is that using COTE will probably give you the most accurate model. If a simpler approach is needed and the discriminatory features are likely to be embedded in subseries, then we would recommend using TSF or ST if the features are in the time domain (depending on whether they are phase dependent or not) or BOSS if they are in the frequency domain. If a whole series elastic measure seems appropriate, then using EE is likely to lead to better predictions than using just DTW.

Finally, we stress that accuracy is not the only consideration when assessing a TSC algorithm. Time and space efficiency are often of equal or greater concern. However, if the only metric used to support a new TSC is accuracy on these test problems, then we believe that evaluation should be transparent and comparable to the results we have made available. If a proposed algorithm is not more accurate than those we have evaluated, then some other case for the algorithm must be made.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) [grant number EP/M015087/1]. The experiments were carried out on the High Performance Computing Cluster supported by the Research and Specialist Computing Support service at the University of East Anglia.

6. REFERENCES

[1] A. Bagnall, A. Bostrom, and J. Lines. The UEA TSC codebase. `https://bitbucket.org/TonyBagnall/time-series-classification`.

[2] A. Bagnall, J. Lines, J. Hills, and A. Bostrom. Time-series classification with COTE: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27:2522–2535, 2015.

[3] A. Bagnall, J. Lines, and E. Keogh. The UCR/UEA Time Series Classification archive. `http://www.timeseriesclassification.com`.

[4] G. Batista, E. Keogh, O. Tataw, and V. deSouza. CID: an efficient complexity-invariant distance measure for time series. *Data Mining and Knowledge Discovery*, 28(3):634–669, 2014.

[5] M. Baydogan and G. Runger. Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30(2):476–509, 2016.

[6] M. Baydogan, G. Runger, and E. Tuv. A bag-of-features framework to classify time series. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):2796–2802, 2013.

[7] A. Bostrom and A. Bagnall. Binary shapelet transform for multiclass time series classification. In *Proc.17th DaWaK*, 2015.

[8] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, 2002.

[9] G. Batista D. Silva, V. de Souza. Time series classification using compression distance of recurrence plots. In *Proc. 13th IEEE ICDM*, 2013.

[10] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[11] H. Deng, G. Runger, E. Tuv, and M. Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[12] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. In *Proc. 34th VLDB*, 2008.

[13] B. Fulcher and N. Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[14] T. Górecki and M. Łuczak. Using derivatives in time series classification. *Data Mining and Knowledge Discovery*, 26(2):310–331, 2013.

[15] T. Górecki and M. Łuczak. Non-isometric transforms in time series classification using DTW. *Knowledge-Based Systems*, 61:98–108, 2014.

[16] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *Proc. 20th SIGKDD*, 2014.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. The WEKA data mining software. *SIGKDD Explorations*, 11(1), 2009.

[18] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[19] Y. Jeong, M. Jeong, and O. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44:2231–2240, 2011.

[20] R. Kate. Using dynamic time warping distances as features for improved time series classification. *Data Mining and Knowledge Discovery*, online first, 2015.

[21] E. Keogh and T. Folias. The UCR time series data mining archive. `http://www.cs.ucr.edu/ẽamonn/time_series_data/`.

[22] J. Lin, E. Keogh, W. Li, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

[23] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012.

[24] J. Lines and A. Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29:565–592, 2015.

[25] P. Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2009.

[26] A. Mueen, E. Keogh, and N. Young. Logical-shapelets: An expressive primitive for time series classification. In *Proc. 17th SIGKDD*, 2011.

[27] T. Rakthanmanon and E. Keogh. Fast-shapelets: A fast algorithm for discovering robust time series shapelets. In *Proc. 13th SDM*, 2013.

[28] C. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *Proc. 5th SDM*, 2005.

[29] J. Rodríguez, C. Alonso, and J. Maestro. Support vector machines of interval-based features for time series classification. *Knowledge-Based Systems*, 18:171–178, 2005.

[30] P. Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[31] P. Senin and S. Malinchik. SAX-VSM: interpretable time series classification using sax and vector space model. In *Proc. 13th IEEE ICDM*, 2013.

[32] A. Stefan, V. Athitsos, and G. Das. The Move-Split-Merge metric for time series. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1425–1438, 2013.

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