SoftwareX 22 (2023) 101366

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

BiometricBlender: Ultra-high dimensional, multi-class synthetic data generator to imitate biometric feature space

Marcell Stippinger [a,*], Dávid Hanák [b], Marcell T. Kurbucz [a,c], Gergely Hanczár [b], Olivér M. Törteli [b], Zoltán Somogyvári [a]

[a] Department of Computational Sciences, Wigner Research Centre for Physics, 29-33 Konkoly-Thege Miklós Street, H-1121 Budapest, Hungary
[b] Cursor Insight, 20-22 Wenlock Road, N1 7GU London, United Kingdom
[c] Institute of Data Analytics and Information Systems, Corvinus University of Budapest, 8 Fővám Square, H-1093, Hungary

ARTICLE INFO

Article history:
Received 9 December 2021
Received in revised form 8 March 2023
Accepted 9 March 2023

Keywords:
Dataset generator
Biometrics
Feature screening
Ultra-high dimensionality
Multi-class classification

ABSTRACT

The lack of freely available (real-life or synthetic) high or ultra-high dimensional, multi-class datasets may hamper the rapidly growing research on feature screening, especially in the field of biometrics, where the usage of such datasets is common. This paper reports a Python package called BiometricBlender, which is an ultra-high dimensional, multi-class synthetic data generator to benchmark a wide range of feature screening methods. During the data generation process, the overall usefulness and the intercorrelations of blended features can be controlled by the user, thus the synthetic feature space is able to imitate the key properties of a real biometric dataset.

Code metadata

Current code version	1.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00238
Code Ocean compute capsule	Not applicable
Legal Code License	MIT License (More information at: https://mit-license.org (retrieved: 22 January 2023).)
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation and installation requirements, operating environments & dependencies	Python 3.7.1+, h5py 2.10+, numpy 1.18+, scipy 1.6+, scikit-learn 0.24+; OS-agnostic (Linux, OS X, MS Windows)
Link to developer documentation and user manual	https://github.com/cursorinsight/biometricblender/blob/paper/README.md
Support email for questions	stippinger.marcell@wigner.hu

Software metadata

Current code version	1.1.0
Permanent link to code/repository used for this code version	https://github.com/cursorinsight/biometricblender/tree/paper
Legal Code License	MIT License (More information at: https://mit-license.org (retrieved: 22 January 2023).)
Compilation and installation requirements, operating environments & dependencies	Python 3.7.1+, h5py 2.10+, numpy 1.18+, scipy 1.6+, scikit-learn 0.24+; OS-agnostic (Linux, OS X, MS Windows)
Link to developer documentation and user manual	https://github.com/cursorinsight/biometricblender/blob/paper/README.md
Support email for questions	stippinger.marcell@wigner.hu

1. Motivation and significance

Analyzing ultra-high dimensional data that include hundreds of thousands of features is becoming an increasingly common problem in many fields of modern scientific research [1]. Since

* Corresponding author.
E-mail address: stippinger.marcell@wigner.hu (Marcell Stippinger).

https://doi.org/10.1016/j.softx.2023.101366

these datasets typically contain only a relatively few relevant, non-redundant predictors, a screening step that removes irrelevant features prior to the main analysis is often employed for reaching a better prediction accuracy and much faster computation [2].

While numerous screening methods have been published in recent years (e.g., [3–9]), only a few high or ultra-high dimensional datasets are available publicly that can be employed for benchmarking purposes. Furthermore, these public datasets (see, e.g., high dimensional datasets related to classification tasks on the UC Irvine Machine Learning Repository[1]) typically do not contain ground truth side information on the usefulness of the features. Besides, most of them have binary response variables, so they cannot be used to benchmark methods developed for solving multiple-class screening problems. While in biometrics such problems are typically encountered, it is difficult to imitate the properties of these kinds of feature spaces by using available data generators (e.g., the Madelon dataset [10] and the associated data generation algorithm implemented by the `make_classification` function of the *scikit-learn* Python package [11]).

To remedy this shortcoming, this paper reports a Python package called BiometricBlender, which is an ultra-high dimensional, multi-class synthetic data generator to benchmark a wide range of feature screening methods. During the data generation process, the overall usefulness and the intercorrelations of features can be controlled by the user. Accordingly, the key properties of a biometric dataset can be imitated by the blended synthetic feature space. This dataset provides an alternative to real biometric datasets, which are typically not freely available. Therefore, it enables the publishing of results achieved on such data.

The paper is organized as follows. Section 2 contains the detailed description of the data generator software. As illustrative examples, Section 3 presents three synthetic feature spaces generated to imitate real-life datasets. Finally, Section 4 summarizes the impact of the software and provides the conclusions.

2. Software description

This section describes the full generator pipeline of BiometricBlender in detail. The output of the pipeline is a high dimensional, multi-class $S \times F^{visible}$ feature matrix $\mathbf{V}^{visible} = \left[v_{ij}^{visible}\right]$, where:

- $F^{visible}$ is the desired number of observable, *visible* features;
- $\mathcal{F}^{visible} = \left\{f_j^{visible} | 1 \le j \le F^{visible}\right\}$ is the set of visible features;
- $C = |\mathcal{C}|$ is the number of classes;
- $S_{\mathcal{C}}$ is the number of samples per class[2]; and
- $S = |\mathcal{S}| = C \cdot S_{\mathcal{C}}$ is the total number of samples.

Visible features are derived from a set of *hidden* features, which are significantly fewer than their visible counterparts. In this context:

- $F^{hidden} \ll F^{visible}$ is the desired number of hidden features;
- $\mathcal{F}^{hidden} = \left\{f_j^{hidden} | 1 \le j \le F^{hidden}\right\}$ is the set of hidden features;
- $\mathcal{F}^{true}$ is the set of hidden features which are created to be significant and distinguishing, and
- $\mathcal{F}^{fake}$ is the set of hidden features which are just pure noise, and do not contribute useful information to the classification of samples. Moreover:

$$\mathcal{F}^{true} \cup \mathcal{F}^{fake} = \mathcal{F}^{hidden},$$
$$\mathcal{F}^{true} \cap \mathcal{F}^{fake} = \emptyset.$$

With an analogy taken from genetics, hidden features are the *genotypes*, visible features are the *phenotypes* of samples. "True feature" genes have an effect on the behavior being observed, while "fake feature" genes do not.

If hidden features were directly observable and ideally distributed, sample classification would be a trivial task. The blender components in the second half of the pipeline (see below) ensure that this information is more concealed in the visible features. The full pipeline performs the following steps:

1. A suitable distribution type and a set of distribution parameters are selected per $c_k \in \mathcal{C}$ class and $f_j^{hidden} \in \mathcal{F}^{hidden}$ hidden feature;
2. F^{hidden} hidden feature values are drawn from these distributions per sample:

$$\forall s_i \in \mathcal{S}, f_j^{hidden} \in \mathcal{F}^{hidden} : v_{ij}^{hidden} = v(s_i, f_j^{hidden});$$

3. Hidden features are combined with each other through polynomial, linear and/or logarithmic combinations to produce $F^{visible}$ visible features per sample;
4. A certain amount of random noise is added to the visible feature values.

2.1. Software architecture

Fig. 1 shows an overview of the pipeline, with the data flow between the components. The number in parentheses after the name of each component indicates which of the above steps the given component participates in. Each component has a number of parameters that control the usefulness and signal-to-noise ratio of the individual features. The components and their parameters are described in Sections 2.1.1 to 2.1.5.

2.1.1. Location factory

This component is responsible for determining the *location*[3] of the distributions per class and hidden feature: $\mathbf{L} = \left[l_{kj} = l(c_k, f_j^{hidden})\right]$. Locations are randomly chosen under an envelope. The envelope is either a *normal distribution* or a *uniform distribution*, specified by the user. The parameters of the Location Factory are:

- **Number of features** (true and fake): F^{true}, F^{fake}.
- **Number of classes:** C.
- **Ordering extent** $\left(\in \mathbb{Z}_{[0,C]}\right)$: controls whether the $l_{kj} : k \in \mathbb{Z}_{[1,C]}$ sequence of locations of any particular f_j^{hidden} feature are randomly, partially or fully ordered, thus controls the correlations between features. Its value specifies the average number of locations in every ordered subsequence. This is relevant when not just one, but several features come into play: the more ordered locations are, the less detail a new feature adds to the overall amount of information.
  Example: *height* and *foot size* are ordered similarly, therefore knowing both does not carry twice as much information as knowing only one of them. *IQ*, on the other hand, is ordered randomly relative to these two features, so knowing both *IQ* and *height* doubles the amount of information.

[1] Available at: https://archive.ics.uci.edu (retrieved: 22 January 2023).

[2] Note that, for simplicity, scalar $S_{\mathcal{C}}$ was used. Further development could provide a more realistic feature space by employing classes with different sample sizes.

[3] *Location* defines the translation of a distribution, e.g., it is the mean of normal distributions, and the smallest value, i.e., the start of the range of values for uniform distributions.

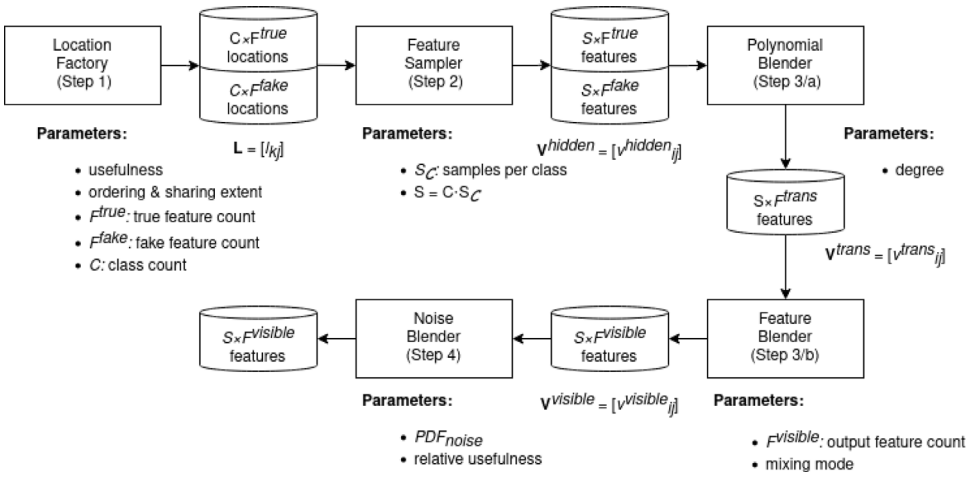

Fig. 1. Main components of the generator pipeline of BiometricBlender.

- **Sharing extent** $\left(\in \mathbb{Z}_{[0,C]}\right)$: controls how many classes share the exact same location on average. With zero sharing extent, all classes have separate, distinguishable locations. With sharing extent C, applied to all fake features, all classes share a single location: $\forall k \in \mathbb{Z}_{[1,C]} : l_{kj} = l_j$, rendering the feature completely useless. A sharing extent in between creates distinguishable groups of classes, within which groups the individual classes appear identical.
  Example: The sharing extent of the *Social Security Number* (SSN) is zero, since all SSNs are unique. The sharing extent of *first names,* on the other hand, is significantly higher.
- **Usefulness** $\left(\in \mathbb{R}_{[0,1]}\right)$ intuitively controls how spread out are the sampling distributions of *Feature Sampler*. The larger the usefulness and the less spread out distributions are, the easier it is to separate feature values generated around these locations. Rather than specifying the usefulness of all hidden features manually, the Location Factory expects a *usefulness scheme*, with which it generates the usefulness of all features. The scheme can be *linear*, *exponential* or *long-tailed*. The usefulness of pure noise features is fixed at zero. Fig. 2 shows different usefulness parameter settings through the example of two hidden features.
  The interpretation of the sampling differences highly depends on the data to be imitated.
  Example 1: When identifying people, the usefulness of SSN is 1, because it never changes, and it is unambiguous. The respiratory rate has much lower, but still non-zero usefulness because while it cannot identify individuals, it can separate some age groups, people doing certain activities, or people with some medical condition that affects breathing.
  Example 2: When measured at short time scales, body weight has very little uncertainty. Finding a difference in body weight implies loss or gain usually achieved over a longer period of time. In contrast, a signature cannot be exactly replicated even under the same conditions, within the same minute.
  The reproducibility of features across different sessions is called persistence in the literature [12,13]. The instantaneous discriminative power is the ability of a feature to distinguish a class from the rest of the population at a given point in time, it can be measured, e.g., by the intraclass correlation coefficient [14]. More generally, the usefulness of a feature depends on both components: persistence and instantaneous discriminative power. In our software, they are modeled via the usefulness parameter, and the number of sessions is defined by the number of samples per class.

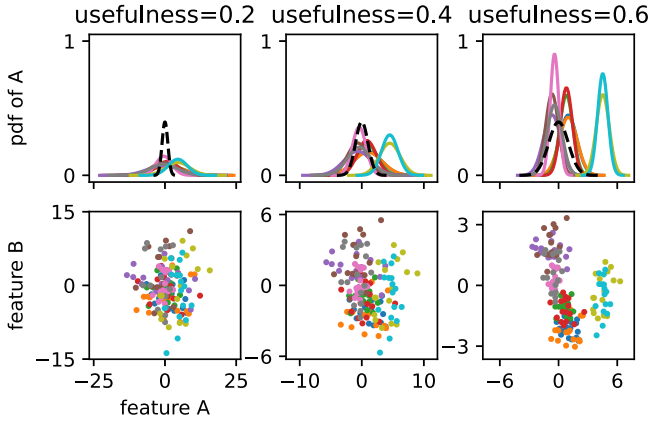

Fig. 2. An example of two hidden features A and B for ten classes with ordering extent 2 and sharing extent 2.

2.1.2. Feature sampler

This component takes the l_{kj} locations and usefulness values of the previous step and draws hidden feature values for the required number of samples from *normal distribution* around these locations. As an option, the *uniform distribution* is also available at the command line interface.

During sampling, the usefulness is converted to the *scale*[4] of the sampling distributions:

- for true features, the converted scale is multiplied by a number drawn from a small uniform distribution around 1, in order to add some variance; and
- for fake features, a fixed scale value is used.

For every class $c_k \in \mathcal{C}$, S_C samples are created, resulting in a $S \times F^{hidden}$ matrix $\mathbf{V}^{hidden} = \left[v_{ij}^{hidden}\right]$ of hidden feature values. Due to the conversion of usefulness to scale, less useful features have larger magnitudes. The magnitudes get normalized just before blending.

2.1.3. Polynomial blender

This component takes all possible combinations of at least one, at most d non-unique hidden features, and multiplies them

[4] *Scale* defines the spread of a distribution, e.g., it is the standard deviation for normal distributions, and the length of the range of values for uniform distributions.

together while preserving their scales by taking the appropriate roots. For example, if feature values are x, y and z, and $d = 2$, then the generated features are x, y, z, $\sqrt{xy}$, $\sqrt{xz}$, $\sqrt{yz}$, $\sqrt{x^2}$, $\sqrt{y^2}$ and $\sqrt{z^2}$.[5]

The output is an $\mathcal{F}^{trans}$ set of $F^{trans} = |\mathcal{F}^{trans}| = \binom{F^{hidden}+d}{d} - 1$ (non-unique) transitional features.[6] The degenerate case of $d = 1$ results in $\mathcal{F}^{trans} = \mathcal{F}^{hidden}$.

2.1.4. Feature blender

This component takes the transitional features, constructs a random, $F^{visible} \times F^{trans}$ dimensional sparse weight matrix $\mathbf{W} = \left[w_{ij}\right]$, and produces $F^{visible}$ blended features using those weights.

The number of blended transitional features per visible feature (i.e., the number of non-zero items in each column of $\mathbf{W}$) is randomly chosen from a discrete uniform distribution of small values. The weights themselves are chosen from a *Dirichlet distribution*, such that their sum per visible feature is always 1: $\forall i : \sum_{j=1}^{F^{trans}} w_{ij} = 1$. Thus, the overall magnitude of the features is preserved during blending. Even so, for classification methods operating with distances, a normalization of visible features may be necessary [15], while other methods, such as the Decision Tree and the Random Forest Classifier, are insensitive to differently scaled features.

The Feature Blender can operate in two modes:

- in *linear* mode, the visible features are weighted sums of the transitional features:

$$\mathbf{V}^{visible} = \left[v_{ij}^{visible}\right] = \mathbf{V}^{trans}\mathbf{W}^{\intercal} \text{ where } v_{ij}^{visible} = \sum_{t=1}^{F^{trans}} v_{it}^{trans} w_{jt};$$

- in *logarithmic* mode, the visible features are products of the weighted powers of the transitional features:

$$\mathbf{V}^{visible} = \left[v_{ij}^{visible}\right] \text{ where } v_{ij}^{visible} = \prod_{t=1}^{F^{trans}} \left(v_{it}^{trans}\right)^{w_{jt}}.$$

The linear mode results in feature distributions close to the Gaussian, while logarithmic mode generates long-tailed feature distributions close to the lognormal distribution.[7] In both modes, the features are expressed as a function of originally uncorrelated mixture distributions (hidden features) consisting of normally distributed components.[8] The correlation of visible features mimics the challenge of redundancy in large feature spaces.

2.1.5. Noise blender

Finally, optional random noise is added to the visible features, taking the following steps per feature:

1. noise is drawn randomly from a normal distribution;
2. an α relative usefulness is drawn randomly from a uniform distribution, by default, between $0 - 1$;
3. the feature and the random noise are blended with either linear or logarithmic interpolation, with α, $1 - \alpha$ weights, respectively.

Thus, if α is 1, zero noise is added, and when α is 0, the random noise completely blocks out the feature values.

2.2. Software functionalities

The sole functionality of BiometricBlender is to generate an ultra-high dimensional, multi-class dataset to benchmark a wide range of feature screening methods. The output is generated as an HDF5 file[9] with the following structure:

- `created_at` (string): timestamp of data generation;
- `features` (dimensions: $S \times F^{visible}$, dimension labels: ["sample", "feature"], attributes map to command line user input or default parameter values as strings): generated visible feature values, i.e., $\mathbf{V}^{visible}$;
- `id` (string): unique identifier (UUID4) created at the data generation;
- `hash` (string): hash of the data or the text "unavailable";
- `labels` (dimension: S, dimension label: "sample"): generated sample labels;
- `names` (dimension: $F^{visible}$, dimension label: "feature"): feature names, i.e., $\mathcal{F}^{visible}$;
- `usefulness` (dimension: $F^{visible}$, dimension label: "feature"): usefulness of features;
- optional `hidden_features` and `hidden_usefulness` describe hidden features similarly to `features` and `usefulness`, respectively.

Given a fixed seed, the output is reproducible up to rounding errors. Note that language-specific implementations and wrappers may present you with different dimensions, e.g., some `R` libraries and `Matlab` expand all arrays to have at least two dimensions.

3. Illustrative examples

3.1. Signature feature space

As the first illustrative example, a synthetic dataset is generated to imitate the private signature feature space of Cursor Insight.[10]

The following custom command line parameters were set:

- `n-classes` = 100;
- `n-samples-per-class` = 16;
- `n-true-features` = 40;
- `n-fake-features` = 160;
- `location-ordering-extent` = 2;
- `location-sharing-extent` = 3;
- `n-features-out` = 10 000;
- `blending-mode` = `'logarithmic'`.

[5] Note that by taking the roots we end up with some repeated values.

[6] F^{trans} is equal to the number of combinations of taking exactly d items of $F^{hidden} + 1$ items—all the hidden features plus the constant 1—at a time, with replacement; minus the sole case of taking 1 d times.

[7] The linear blending mode (optionally with the noise blending, see below) conforms to the generative model of the *Factor Analysis* (FA), thus, in theory, the hidden features may be reconstructed up to a multidimensional rotation and some noise. The logarithmic blending mode is different, but the FA still produced reasonable reconstructions.

[8] All correlation between them is due to the realization of individual class means. Although it is tempting to handle the problem of a large number of visible features by decorrelating them, e.g., by using the Cholesky transformation [13], the algebraic solution can be unstable due to a large number of visible features in the linear case. Moreover, for the logarithmic case, such a transformation does not return the original independent features, just some dependent variables with diagonal variance.

[9] Detailed information about the HDF5 format can be found at: https://hdfgroup.org/solutions/hdf5 (retrieved: 22 January 2023).

[10] Cursor Insight won the ICDAR competition on signature verification and writer identification in 2015 [16]. For further information, see: https://cursorinsight.com/e-signatures.html (retrieved: 22 January 2023). Note that, to demonstrate the potential in screening, the dataset generated here is somewhat noisier than the imitated data.

Table 1
Classification results on the 1600 × 10 000 dataset for three basic classifiers and various reduction algorithms. *(a)* Only the best accuracy among all parameters is reported. *(b)* Fit times are the wall time after the reduction step and correspond to the accuracy shown above.

(a) Classification performance

Reduction		None	PCA	FA	*k*-best	$\mathcal{F}^{true}$
	*k*NN	0.131	0.218	0.214	0.641	0.632
Class.	SVC	0.471	0.466	0.548	0.686	0.656
	RF	0.609	0.371	0.716	0.692	0.860

(b) Fit time of the classifier

Reduction		None	PCA	FA	*k*-best
	*k*NN	0.153 s	0.003 s	0.001 s	0.006 s
Class.	SVC	24 s	0.37 s	0.42 s	0.46 s
	RF	300 s	22 s	21 s	29 s

The resulting dataset has 1600 samples and 10 000 features. Note that in this feature set one must adjust the parameters of the generative model to approximate the statistics (e.g., eigenspectrum) of the output rather than prescribing the statistics themselves. We tested it for classification in the following ways. We trained the *scikit-learn* [11] (version: 0.24.2) implementation of three basic classifiers on the original data and on the reduced/decomposed version of the data. We characterized the best cross-validated accuracy that can be attained for each classifier using a full grid search over crucial parameters. These parameters were:

- `weights = 'uniform', 'distance'` for *k*-nearest neighbors (*k*NN);
- `C` = 0.5, 1.0, 2.0; `tol` = 1e-4, 1e-3, 1e-2 for the support vector classifier (SVC); and
- `n_estimators` = 1000; `min_samples_leaf` = 1, 2, 4; `max_depth` = None, 8, 10; `min_impurity_decrease` = 0.0, 0.01, 0.05 for the Random Forest Classifier (RF).

The reduction step allowed the classifiers to work on a more focused dataset. We executed each reduction/decomposition algorithm to produce a reduced feature space of 10, 25, 50, 100, 200, 400, and 800 features and reported the best accuracy only, see Table 1. The Principal Component Analysis (PCA) kept its default settings. To Factor Analysis (FA) we applied the `varimax` rotation. The *k*-best `SelectKBest` method increasingly selected the best features using the `f_classif` score. $\mathcal{F}^{true}$ used the true hidden features.

3.2. FaceNet feature space

The deep neural network FaceNet [17] provides 128 features for each portrait presented. Those features are approximately normal with little redundancy (low average correlations) and insensitive to irrelevant features, such as lighting (low noise). On the MORPH-II dataset [18], PCA has been reported to explain the feature variance with half of the total features [19]. Therefore, we assume that 64 true and no fake features are required for producing the dataset. We can expose the true features by decreasing the number of true features blended in each visible feature and at the same time reducing the observation noise.

The following custom command line parameters are recommended:

- `n-true-features` = 64;
- `n-fake-features` = 0;
- `n-features-out` = 128;
- `min-count` = 2;
- `max-count` = 3;
- `max-noise` = 0.2.

The expectations are met, see Fig. 3.

3.3. Vocalization database

For the purpose of studying Parkinson's Disease markers, vocal samples were collected from 252 subjects [20]. PCA indicates that 168 features can explain 95% of the variance. Unlike in the previous case, due to the shape of the feature space (756 samples, 753 features), one cannot tell how many of the features are true without actually solving the classification problem. But our framework is capable of simulating such situations.

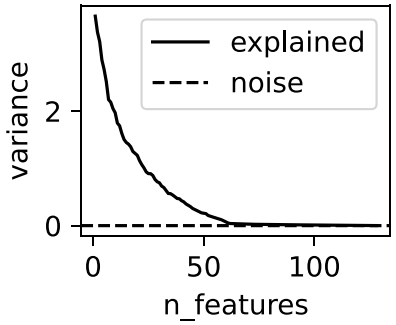

Fig. 3. Variance explained by PCA components of the feature space generated to mimic the FaceNet dataset.

Sakar et al. [20] demonstrated that 50 non-redundant features are required for solving the classification task with 86% accuracy. Based on their scenario, we hypothesize 50 true and 118 fake features, thus the following custom command line parameters are recommended:

- `n-classes` = 252;
- `n-samples-per-class` = 3;
- `n-true-features` = 50;
- `n-fake-features` = 118;
- `n-features-out` = 753;
- `min-count` = 2;
- `max-count` = 4;
- `max-noise` = 0.3.

3.4. Gallery size

To test the effect of gallery size, one usually wants to generate a large number of classes, then feed the filtered output to the algorithm being investigated. Note that the data must be generated in one step. Due to implementation details, generating a smaller number of classes, i.e., 1000 classes does not result in a subset of the synthetic data generated requesting 10 000 features, even if the seed is the same.

The testing shall go on as described in the literature. This example shows only, the parameters of generating a dataset with many classes as follows:

- `n-classes` = 10 000;
- `n-samples-per-class` = 4;
- `n-true-features` = 50;
- `n-features-out` = 2000.

In addition, one can experiment with setting more parameters, such as `min-usefulness`, `max-usefulness`, `location-ordering-extent` or `location-sharing-extent`, see the program usage on GitHub [21].

4. Impact and conclusion

The ultra-high dimensional, multi-class data generator called BiometricBlender supports the rapidly growing research on feature screening in two ways. On the one hand, it facilitates the benchmark of a wide range of feature screening methods (see Table 1) by providing an alternative to real (typically non-free) biometric datasets. On the other hand, it enables the publishing of results achieved on such data. To this end, the overall usefulness and the intercorrelations of blended features can be controlled by the user during data generation. In addition, it is possible to study how the classification performance changes with the number of classes (gallery size) [19], with the ratio of true and fake features, their overall number, or with their mixing (blending parameters). Thus, the synthetic feature space is able to imitate the key properties of a real biometric dataset.

We wish to draw the attention of the reader to the following facts, which may be considered as potential conflicts of interest, and to significant financial contributions to this work. The nature of the potential conflict of interest is described below: some of the authors work for Cursor Insight, an IT company targeting human motion analysis, person classification, and identification based on large-scale biometric data in particular. In order to handle such real-life, multi-class, ultra-high dimension datasets efficiently, we came up with our own feature screening algorithm, because we found industry-standard solutions insufficient. We have then decided to share our solution publicly. The demand for a synthetic data generator arose when, in order to prove the performance of our screening algorithm against standard solutions, we started looking for publicly available reference datasets of such dimensions, or generators of such, and found none.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: We wish to draw the attention of the Editor to the following facts, which may be considered as potential conflicts of interest, and to significant financial contributions to this work. The nature of potential conflict of interest is described below: some of the authors work for Cursor Insight, an IT company targeting human motion analysis, person classification and identification based on large-scale biometric data in particular. In order to handle such real-life, multi-class, ultra-high dimension datasets efficiently, we came up with our own feature screening algorithm, because we found industry standard solutions insufficient. We have then decided to share our solution with the general public. The demand for a synthetic data generator arose when, in order to prove the performance of our screening algorithm against standard solutions, we started looking for publicly available reference datasets of such dimensions, or generators of such, and found none.

Data availability

No data was used for the research described in the article.

Acknowledgments

The authors would like to thank Erika Griechisch (Cursor Insight, London) and András Telcs (Wigner Research Centre for Physics, Budapest) for their valuable comments and advice. M.S., M.T.K., and Z.S. thank the support of Eötvös Loránd Research Network, Hungary (Z.S. was supported under grant number SA-114/2021). Z.S. was supported by the Hungarian National Research, Development and Innovation Office, under grant number NKFIH K135837 and the Hungarian National Brain Research Program 2017-1.2.1-NKP-2017-00002. M.T.K. received support from the Ministry of Innovation and Technology NRDI Office, Hungary within the framework of the MILAB Artificial Intelligence National Laboratory Program, and from the Hungarian Scientific Research Fund (OTKA/NKFIH) under contract number PD142593.

References

[1] Liu Y, Chen X. Quantile screening for ultra-high-dimensional heterogeneous data conditional on some variables. J Stat Comput Simul 2018;88(2):329–42. http://dx.doi.org/10.1214/13-AOS1087.
[2] Qiu D, Ahn J. Grouped variable screening for ultra-high dimensional data for linear model. Comput Statist Data Anal 2020;144:106894. http://dx.doi.org/10.1016/j.csda.2019.106894.
[3] Mai Q, Zou H. The Kolmogorov filter for variable screening in high-dimensional binary classification. Biometrika 2013;100(1):229–34. http://dx.doi.org/10.1093/biomet/ass062.
[4] Mai Q, Zou H, et al. The fused Kolmogorov filter: A nonparametric model-free screening method. Ann Statist 2015;43(4):1471–97. http://dx.doi.org/10.1214/14-AOS1303.
[5] Chen X, Chen X, Wang H. Robust feature screening for ultra-high dimensional right censored data via distance correlation. Comput Statist Data Anal 2018;119:118–38. http://dx.doi.org/10.1016/j.csda.2017.10.004.
[6] Chen X, Zhang Y, Liu Y, Chen X. Model-free feature screening for ultra-high dimensional competing risks data. Statist Probab Lett 2020;164:108815. http://dx.doi.org/10.1016/j.spl.2020.108815.
[7] Chen X, Liu CC, Xu S. An efficient algorithm for joint feature screening in ultrahigh-dimensional Cox's model. Comput Statist 2021;36(2):885–910. http://dx.doi.org/10.1007/s00180-020-01032-9.
[8] He Y, Zhang L, Ji J, Zhang X. Robust feature screening for elliptical copula regression model. J Multivariate Anal 2019;173:568–82. http://dx.doi.org/10.1016/j.jmva.2019.05.003.
[9] Hu Q, Lin L. Feature screening in high dimensional regression with endogenous covariates. Comput Econ 2021;1–21. http://dx.doi.org/10.1007/s10614-021-10174-x.
[10] Guyon I, Gunn S, Ben-Hur A, Dror G. Result analysis of the NIPS 2003 feature selection challenge. Adv Neural Inf Process Syst 2004;17.
[11] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30. http://dx.doi.org/10.48550/arXiv.1201.0490.
[12] Friedman L, Nixon MS, Komogortsev OV. Method to assess the temporal persistence of potential biometric features: Application to oculomotor, gait, face and brain structure databases. PLoS One 2017;12(6):1–42. http://dx.doi.org/10.1371/journal.pone.0178501.
[13] Friedman L, Stern HS, Price LR, Komogortsev OV. Why temporal persistence of biometric features, as assessed by the intraclass correlation coefficient, is so valuable for classification performance. Sensors 2020;20(16). http://dx.doi.org/10.3390/s20164555.
[14] Fisher R. Statistical methods for research workers. Edinburgh: Oliver and Boyd; 1925.
[15] Friedman L, Komogortsev OV. Assessment of the effectiveness of seven biometric feature normalization techniques. IEEE Trans Inf Forensics Secur 2019;14(10):2528–36. http://dx.doi.org/10.1109/TIFS.2019.2904844.
[16] Malik MI, Ahmed S, Marcelli A, Pal U, Blumenstein M, Alewijns L, Liwicki M. ICDAR2015 competition on signature verification and writer identification for on-and off-line skilled forgeries (SigWIcomp2015). In: 2015 13th International Conference on Document Analysis and Recognition. ICDAR, IEEE; 2015, p. 1186–90. http://dx.doi.org/10.1109/ICDAR.2015.7333948.
[17] Schroff F, Kalenichenko D, Philbin J. FaceNet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015, p. 815–23. http://dx.doi.org/10.1109/CVPR.2015.7298682.
[18] Ricanek K, Tesafaye T. MORPH: A longitudinal image database of normal adult age-progression. In: 7th International conference on automatic face and gesture recognition (FGR06). IEEE; 2006, p. 341–5. http://dx.doi.org/10.1109/FGR.2006.78.
[19] Friedman L, Stern H, Prokopenko V, Djanian S, Griffith H, Komogortsev O. Biometric performance as a function of gallery size. Appl Sci 2022;12(21). http://dx.doi.org/10.3390/app122111144.
[20] Sakar CO, Serbes G, Gunduz A, Tunc HC, Nizam H, Sakar BE, Tutuncu M, Aydin T, Isenkul ME, Apaydin H. A comparative analysis of speech signal processing algorithms for Parkinson's disease classification and the use of the tunable Q-factor wavelet transform. Appl Soft Comput 2019;74:255–63. http://dx.doi.org/10.1016/j.asoc.2018.10.022.
[21] Stippinger M, Hanák D, Kurbucz MT, Hanczár G, Törteli OM, Hergert L, Somogyvári Z. BiometricBlender. 2022, GitHub URL https://github.com/cursorinsight/biometricblender.