NeurIPS Hide-and-seek Privacy Challenge documentation questionnaire

Team name

PrivacyHEL

Submission filenames(s)

Hider	hider.py
	generative_svd.py (main model) sparse_feature_prediction.py (sparse model) data_pipeline*.py post_process.py
Seeker	

What class of algorithms does your solution belong to? (e.g. GANs, VAEs, noise-injection, nearest neighbor, etc.)

Hider	(Generative) matrix decomposition
Seeker	

Describe your algorithm in one sentence (e.g. "Noise is added to the original data and then this data is returned.")

Hider	After preprocessing, run main model: transform timeseries into a trajectory matrix, do SVD, throw out some components + optionally add noise to r.s.v., optionally learn a (noisy) generative model on l.s.v., reconstruct trajectory, transform back into timeseries.
Seeker	

Describe your algorithm in words (e.g. "Noise is drawn from a Gaussian distribution, with mean 0 and variance s, where the dimension is determined by the size of the dataset. This noise is added to the original data to produce a noisy version of the dataset and this noisy dataset is then returned as the synthetic data.")

Hider	Do various preprocessing steps, some of which will be inverted in postprocessing. Use one of 3 approaches for modelling based on feature properties: Main model: transform timeseries into trajectory matrix and do SVD on that. Discard some components. The final version skips all noise additions and generative modelling. Transform back into timeseries. Auxiliary sparse model: for sparse features, train a simple neural network to predict value at each timestep based on other features. For very sparse features and those consistently performing poorly in evaluation: when there's a closely related feature with good utility available replace the bad feature with the related feature, otherwise use original data with small amount of Gaussian noise perturbation.
Seeker	

Specify any loss functions used (e.g. "No loss functions used.")

Hider	No loss for main model, MSE for sparse feature model
Seeker	

Specify any hyperparameters and how they are optimized (or preset values) (e.g. "The noise size, s, is set to 0.1.")

Hider	All hyperparameters have been set by optimizing performance in our local testing. Most of the hyperparameters in the main hider model are not relevant for the final submitted model (the values used will basically skip the corresponding parts of the model). The remaining parameters tune the number of components to keep from SVD and the sliding window length for building a trajectory matrix. The sparse feature prediction network is trained for a number of epochs that led to convergence of the loss in local tests, plus some buffer. Hyperparameters of the network have been empirically determined as a compromise between fidelity and training cost. We also use some explicit (loose) bounds on the data in postprocessing the generated data. These are based on subject domain knowledge (physiological quantities and measurements tend to have some upper & lower bounds for living people). We also replace some generated features with physiologically closely related other features, that are easier to model.
Seeker	

Specify any pre-trained models used by your algorithm (e.g. "None.")

Hider	None
Seeker	

Pseudo-code for your algorithm

e.g. **Inputs:** Dataset, D, random seed
Hyperparameters: s (default 0.1)

1. Determine dataset dimension: n x d x T
2. Draw N ~ N(0, s), an n x d x T dimensional Gaussian
3. Return D + N

Hider	**General algorithm** (see separate descriptions for each part below) 1. Preprocess raw data 2. Divide features into 3 sets: i) main model, ii) sparse model, iii) bad features 3. Generate synthetic data for a given feature from the corresponding model 4. Postprocess generated data Main model is used for modelling "normal" features, which have enough samples and where the results are not consistently bad in terms of utility. **Main model**: 1. Enforce correct Pads+nans (safeguard, should already be treated by preprocessing) 2. Transform timeseries D into trajectory matrix M 3. Do SVD: U,s,V = SVD(M,k), keeping only a given number k of components 4. Reconstruct M' = U s V 5. Transform reconstructed trajectory matrix M' into timeseries D' 6. Perturb first non-padded observation of each feature in D' very slightly to avoid triggering competition evaluation in classification mode 7. Return perturbed D' as synthetic data Sparse model is used for modelling features that have only a few samples (less than 10% frequency per time series on average). We predict these from features that are dense (> 20% frequency per time series on average). **Sparse model**: 1. Create training set by isolating time steps with observations for at least one sparse feature and corresponding windows of dense features centered at that step. 2. Train neural network to predict sparse feature values from windows of dense feature values (on original pre-processed data), minimizing MSE for sparse feature predictions. 3. Predict synthetic values for sparse features using the trained neural network on non-sparse features generated by main model. Bad features are features that have consistently failed our utility tests. **Bad features** (selected based on having consistently bad utility scores in our tests): For features where we have a closely related feature with better utility results available: replace the bad feature with the related good feature. For other features: use original data with a small Gaussian perturbation (input perturbation). Preprocessing pipeline is applied to the raw data. The processed data is input to the main model. Sparse model and bad feature inputs are forked from the pipeline at appropriate steps. **Preprocessing**:

	1. Change padding mask into 2D 2. Enforce finiteness by replacing infinite with nans 3. Flip padding to the end of the series 4. Clip values to (.05, .95)-quantiles 5. Remove features which can be derived with simple transformations from other features 6. Log-scale data 7. Do competition imputation 8. Std-scale data 9. Remove features which don't have any observations (constant 0s) Postprocessing is applied to the synthetic data generated by the main model (as well as to the sparse model and bad features output when this makes sense). It inverts the steps done by preprocessing in reverse order, except for the steps specified below (numbering corresponds to preprocessing steps) **Postprocessing**: 9. Return all-0 features to data 7. No action 5. Return features by transforming from the generated features 4. Clip generated values to pre-specified bounds (based on prior knowledge) 2. Enforce finiteness by replacing infinite with nans
Seeker	

Finally, alongside this document **please also submit a commented version of your code**. Please include:
- Docstrings for each new class/function defined
- Inline comments for your main function/class

The goal of these comments is to tie the code to the description you have provided here. Please do not alter the actual content of your code - only add comments/docstrings.

Submitting your documentation and commented code

Please submit your commented code within a .zip or equivalent file type (1 file per solution), and share it with us as an attachment alongside this Word doc.

You can send these via email (to nm736@cam.ac.uk; james.jordon@wolfson.ox.ac.uk; es583@cam.ac.uk) or DM James Jordon/Evgeny Saveliev on Slack (you can join the workspace with this URL).