wgan

Release 0.1

Evan Munro and Jonas Metzger

Nov 21, 2022

CONTENTS:

In the following example we are interested in fitting the distribution of covariates conditional on a treatment. The data set `df` includes covariates and the context variable, the treatment. Note that in some applications where the original dataset is very unbalanced, for best results it may be necessary to balance the dataset before training with respect to context variable.

To set up the *DataWrapper*, we need to categorize the variables in `df` in the following way:

- `continuous_vars`: list of continuous variables to be generated
- `continuous_lower_bounds`: define lower bound of continuous variables (if applicable)
- `continuous_upper_bounds`: define upper bound of continuous variables (if applicable)
- `categorical_vars`: list of categorical variables to be generated
- `context_vars`: list of variables that are conditioned on when generating data (cWGAN)

The default value for each of these settings is an empty list, but at least one of `continuous_vars` and `categorical_vars` must be non-empty when setting up a *DataWrapper*.

```
continuous_vars = ["continuous_var_1", "continuous_var_2"]
continuous_lower_bounds = {"continuous_var_1": 0}
categorical_vars = ["categorical_var"]
context_vars = ["t"]
data_wrapper = wgan.DataWrapper(df, continuous_vars, categorical_vars,
                                context_vars, continuous_lower_bound)
```

DataWrapper prepares the data in `df`. Before the training of the *Generator* and *Critic*, `df` is scaled using the function *preprocess*. After the training procedure, generated data is rescaled to the original data set.

```
x, context = data_wrapper.preprocess(df)
```

If `context_vars` is an empty list, then *preprocess* will return an empty `context`. *Specifications* specifies the tuning parameters for the training process based on a *DataWrapper* before training the *Generator* and *Critic*. The resulting object `specs` includes all the tuning parameters for the training process.

We include some suggested guidelines for the tuning parameters that we find need adjusting from the default values most frequently. Training GANs is not always easy, so some experimentation is likely necessary with a new dataset before getting good results for the generated data. For a dataset with N observations and p covariates:

- `batch_size` should be a fraction of N, we found between 0.1 and 0.5 tends to work best
- `max_epochs` is dataset specific: smaller N tends to require larger *max_epochs*
- `critic_d_hidden` and `generator_d_hidden` should have larger widths for larger p

```
specs = wgan.Specifications(data_wrapper, batch_size=2048, max_epochs=600)
```

Generator is the generator in the WGAN setup and generates new observations based on the distributions in the data set `df`. The underlying function is a dense neural network. The only input required are the specifications `specs`.

```
generator = wgan.Generator(specs)
```

Critic is the discriminator in the WGAN setup and classifies observations as coming from `df` rather than from the *Generator*. The underlying function is a dense neural network.

```
critic = wgan.Critic(specs)
```

See the classes *DataWrapper*, *Specifications*, *Generator*, *Critic* in the *API* for more details, including additional tuning parameters for advanced users.

CHAPTER

ONE

TRAINING MODELS AND GENERATING ARTIFICIAL DATA

1.1 Training Models

The function *train* trains the WGAN model, which is made up of the *Generator* and *Critic* (discriminator). If `context` is non-empty, a cWGAN is trained, otherwise the default is a regular WGAN. The function is trained using stochastic optimization as described in detail in Gulrajani et al 2017.

```
wgan.train(generator, critic, x, context, specs)
```

1.2 Generating Artificial Data

The function `apply_generator` from class *DataWrapper* replaces columns in `df` that are produced by the generator. The generated data is of size equal to the number of rows in `df`. Variables in `df` that are not produced by the generator are not modified.

`df_generated` contains the artificially generated data.

```
df_generated = data_wrapper.apply_generator(generator, df.sample(int(1e6),␣
↪replace=True))
```

CHAPTER

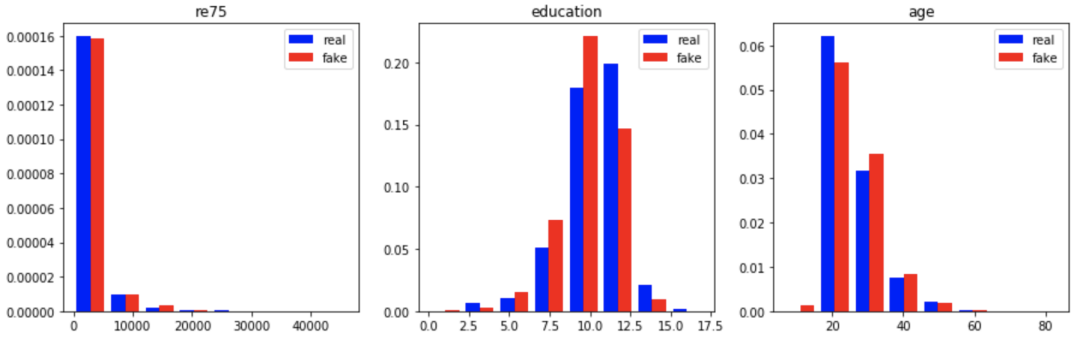
TWO

EVALUATING MODELS

The function *compare_dfs* compares the real data set `df` and the artificially generated data set `df_generated` from the WGANs. The output are a comparison of means, comparisons of standard deviations, histograms and scatterplots to evaluate the performance of the WGAN.

```
wgan.compare_dfs(df, df_generated,
                 scatterplot=dict(x=["continuous_var_1"], y=["continuous_var_2"],
                 samples=400, smooth=0),
                 table_groupby=["context_var"],
                 histogram=dict(variables=["continuous_var_1", "continuous_var_2"],
                 nrow=2, ncol=2),
                 figsize=3)
```

Find below an example of a histogram produced by *compare_dfs*, from Athey et al. [2019].



The figure shows the histograms for CPS data, Earnings 1975, education and age (Figure 3 in the paper). *real* refers to the distribution of the variable in the original data set `df` and *fake* refers to the distribution of the same variable in the artificially generated data set `df_generated`.

See the function *compare_dfs* in the *API* for more details.

CHAPTER

THREE

API

3.1 DataWrapper

class wgan.**DataWrapper**(*df*, *continuous_vars=[]*, *categorical_vars=[]*, *context_vars=[]*, *continuous_lower_bounds=dict()*, *continuous_upper_bounds=dict()*)

Class for processing raw training data for training Wasserstein GAN

Parameters

- **df** (*pandas.DataFrame*) – Training data frame, includes both variables to be generated, and variables to be conditioned on
- **continuous_vars** (*list*) – List of str of continuous variables to be generated
- **categorical_vars** (*list*) – List of str of categorical variables to be generated
- **context_vars** (*list*) – List of str of variables that are conditioned on for cWGAN
- **continuous_lower_bounds** (*dict*) – Key is element of continuous_vars, value is lower limit on that variable.
- **continuous_upper_bounds** (*dict*) – Key is element of continuous_vars, value is upper limit on that variable.

apply_generator(*generator*, *df*)

Replaces or inserts columns in DataFrame that are generated by the generator, of size equal to the number of rows in the DataFrame that is passed

Parameters

- **df** (*pandas.DataFrame*) – Must contain columns listed in self.variables["context"], which the generator will be conditioned on. Even without context vars, len(df) is used to infer the desired sample size, so you need to supply at least pd.DataFrame(index=range(n))
- **generator** (*wgan_model.Generator*) – Trained generator for simulating data

Returns
Original DataFrame with columns replaced by generated data where possible.

Return type
pandas.DataFrame

deprocess(*x*, *context*)

Unscale tensors from WGAN output to original scale

Parameters

- **x** (*torch.tensor*) – Generated data

- **context** (*torch.tensor*) – Data conditioned on

Returns
df – DataFrame with data converted back to original scale

Return type
pandas.DataFrame

preprocess(*df*)

Scale training data for training in WGANs

Parameters
df (*pandas.DataFrame*) – raw training data

Returns

- **x** (*torch.tensor*) – training data to be generated by WGAN
- **context** (*torch.tensor*) – training data to be conditioned on by WGAN

3.2 Specifications

class wgan.**Specifications**(*data_wrapper, optimizer=torch.optim.Adam, critic_d_hidden=[128, 128, 128], critic_dropout=0, critic_steps=15, critic_lr=0.0001, critic_gp_factor=5, generator_d_hidden=[128, 128, 128], generator_dropout=0.1, generator_lr=0.0001, generator_d_noise='generator_d_output', generator_optimizer='optimizer', max_epochs=1000, batch_size=32, test_set_size=16, load_checkpoint=None, save_checkpoint=None, save_every=100, print_every=200, device='cuda' if torch.cuda.is_available() else 'cpu'*)

Class used to set up WGAN training specifications before training Generator and Critic.

Parameters

- **data_wrapper** (*wgan_model.DataWrapper*) – Object containing details on data frame to be trained
- **optimizer** (*torch.optim.Optimizer*) – The torch.optim.Optimizer object used for training the networks, per default torch.optim.Adam
- **critic_d_hidden** (*list*) – List of int, length equal to the number of hidden layers in the critic, giving the size of each hidden layer.
- **critic_dropout** (*float*) – Dropout parameter for critic (see Srivastava et al 2014)
- **critic_steps** (*int*) – Number of critic training steps taken for each generator training step
- **critic_lr** (*float*) – Initial learning rate for critic
- **critic_gp_factor** (*float*) – Weight on gradient penalty for critic loss function
- **generator_d_hidden** (*list*) – List of int, length equal to the number of hidden layers in generator, giving the size of each hidden layer.
- **generator_dropout** (*float*) – Dropout parameter for generator (See Srivastava et al 2014)
- **generator_lr** (*float*) – Initial learning rate for generator
- **generator_d_noise** (*int*) – The dimension of the noise input to the generator. Default sets to the output dimension of the generator.

- **generator_optimizer** (*torch.optim.Optimizer*) – The torch.optim.Optimizer object used for training the generator network if different from "optimizer", per default the same
- **max_epochs** (*int*) – The number of times to train the network on the whole dataset.
- **batch_size** (*int*) – The batch size for each training iteration.
- **test_set_size** (*int*) – Holdout test set for calculating out of sample wasserstein distance.
- **load_checkpoint** (*str*) – Filepath to existing model weights to start training from.
- **save_checkpoint** (*str*) – Filepath of folder to save model weights every save_every iterations
- **save_every** (*int*) – If save_checkpoint is not None, then how often to save checkpoint of model weights during training.
- **print_every** (*int*) – How often to print training status during training.
- **device** (*str*) – Either "cuda" if GPU is available or "cpu" if not

3.3 Generator

class wgan.**Generator**(*specifications*)

torch.nn.Module class for generator network in WGAN

Parameters

specifications (*wgan_model.Specifications*) – parameters for training WGAN

3.4 Critic

class wgan.**Critic**(*specifications*)

torch.nn.Module for critic in WGAN framework

Parameters

specifications (*wgan_model.Specifications*) –

3.5 train

wgan.**train**(*generator*, *critic*, *x*, *context*, *specifications*, *penalty=None*)

Function for training generator and critic in conditional WGAN-GP If context is empty, trains a regular WGAN-GP. See Gulrajani et al 2017 for details on training procedure.

Parameters

- **generator** (*wgan_model.Generator*) – Generator network to be trained
- **critic** (*wgan_model.Critic*) – Critic network to be trained
- **x** (*torch.tensor*) – Training data for generated data
- **context** (*torch.tensor*) – Data conditioned on for generating data
- **specifications** (*wgan_model.Specifications*) – Includes all the tuning parameters for training

3.6 compare_dfs

wgan.**compare_dfs**(*df_real*, *df_fake*, *scatterplot=dict(x=[], y=[], samples=400, smooth=0)*, *table_groupby=[]*, *histogram=dict(variables=[], nrow=1, ncol=1)*, *figsize=3*, *save=False*, *path=''*)

Diagnostic function for comparing real and generated data from WGAN models. Prints out comparison of means, comparisons of standard deviations, and histograms and scatterplots.

Parameters

- **df_real** (*pandas.DataFrame*) – real data
- **df_fake** (*pandas.DataFrame*) – data produced by generator
- **scatterplot** (*dict*) – Contains specifications for plotting scatterplots of variables in real and fake data
- **table_groupby** (*list*) – List of variables to group mean and standard deviation table by
- **histogram** (*dict*) – Contains specifications for plotting histograms comparing marginal densities of real and fake data
- **save** (*bool*) – Indicate whether to save results to file or print them
- **path** (*string*) – Path to save diagnostics for model

CHAPTER

FOUR

INTRODUCTION

wgan is a python module built on top of PyTorch for using Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) to simulate data with a known ground truth from real datasets, in order to test the properties of different estimators, as described in Athey et al. [2019]. The module contains functionality to simulate from either joint or conditional distributions. This documentation will explain how to set up the data, train the models, generate the artificial data and evaluate the models.

Generative Adversarial Networks (GANs) consist of two parts, the generator and a discriminator. The generator generates new observations that look similar to training data by maximizing the probability that the discriminator makes a mistake; the discriminator minimizes the probability of misclassifying generated data as real data. In the wgan module both the generator and the discriminator are neural networks.

The workflow for fitting a distribution and generating data from it using the module is as follows:

1. *Setting up Data and Models*
    - Load data into memory
    - Initialize a *DataWrapper* object and specify the data type for each variable
    - Initialize *Specifications* object given the *DataWrapper*, which specifies hyperparameters for training
    - Initialize *Generator* (generator) & *Critic* (discriminator) given the *Specifications*
    - Normalize the data with the *DataWrapper* object
2. *Training Models*:
    - Train the *Generator* & *Critic* via *train*
3. *Generating Artificial Data*
    - Replace columns in df with simulated data from *Generator* using *DataWrapper.apply_generator*
4. *Evaluating Models*:
    - Check the generated data via *compare_dfs*
    - Save generated data

For bug reports and feature requests, please submit an issue in the Github repository. The repository also contains a Google Colab tutorial that can be accessed here

PYTHON MODULE INDEX

INDEX