Synthesizing Property & Casualty Ratemaking Datasets using Generative Adversarial Networks

Marie-Pier Côté[a], Brian Hartman[b,c], Olivier Mercier[a], Joshua Meyers[b], Jared Cummings[b], Elijah Harmon[b]

[a]*École d'actuariat, Université Laval, Québec, QC, Canada*
[b]*Department of Statistics, Brigham Young University, Provo, UT, USA*
[c]*Corresponding Author: hartman@stat.byu.edu*

Abstract

Due to confidentiality issues, it can be difficult to access or share interesting datasets for methodological development in actuarial science, or other fields where personal data are important. We show how to design three different types of generative adversarial networks (GANs) that can build a synthetic insurance dataset from a confidential original dataset. The goal is to obtain synthetic data that no longer contains sensitive information but still has the same structure as the original dataset and retains the multivariate relationships. In order to adequately model the specific characteristics of insurance data, we use GAN architectures adapted for multi-categorical data: a Wassertein GAN with gradient penalty (MC-WGAN-GP), a conditional tabular GAN (CTGAN) and a Mixed Numerical and Categorical Differentially Private GAN (MNCDP-GAN). For transparency, the approaches are illustrated using a public dataset, the French motor third party liability data. We compare the three different GANs on various aspects: ability to reproduce the original data structure and predictive models, privacy, and ease of use. We find that the MC-WGAN-GP synthesizes the best data, the CTGAN is the easiest to use, and the MNCDP-GAN guarantees differential privacy.

1. Introduction

In order to improve the quality and accuracy of the models used in insurance practice, methodological developments must be tested on the type of data they are meant to model. Unfortunately, insurance claims data at the individual policyholder or claimant level are highly confidential. Just like medical records, these data cannot be publicly shared unless meaningful covariates are erased. This lack of publicly available data slows down the methodological developments in actuarial science.

Take as an example loss reserving. With the improved availability of computing resources, reserving methods that traditionally use aggregate information may now model individual claims. To this end, Antonio and Plat (2014), Pigeon et al. (2013, 2014), and Wüthrich (2018a) proposed micro-level reserving models and illustrated their efficacy on confidential datasets. Because the data is confidential, it is difficult to compare the methods to each other or to new methods yet to be developed. Additionally, the research is not easily reproducible, even when the code is shared.

The lack of publicly available data was discussed in Gabrielli and Wüthrich (2018), where the authors provide an R program for simulating insurance claim development patterns. A Gaussian copula with appropriate margins generates the features, and the different parts of the development process are modeled with successive neural nets. The simulation machine accommodates only a few covariates; the generation of a large number of features with the Gaussian copula could lead to unrealistic combinations of factor levels. In this paper, we propose to synthesize insurance data with a generative adversarial network (GAN).

A GAN is a deep learning model that was introduced in Goodfellow et al. (2014). It consists of two competing neural networks: one that generates fake data, the so-called generator, and a second, the discriminator, that is trained to identify whether the data is real or fake. During the training process, the generator adapts in order to fool the discriminator, which means that it learns to generate fake data that is indistinguishable from the real data. The resulting GAN could thus be used to simulate a synthetic dataset, that is completely fake, but still has the structure of real data.

Frid-Adar et al. (2018) use GANs to generate synthetic data in order to augment a small imaging dataset and improve the performance of the classification of liver lesion. As explained in Papernot et al. (2017), a method based on GANs can provide strong privacy for sensitive training data. Choi et al. (2017) proposed the medGAN architecture to synthesize realistic patient records. Their motivation is similar to ours: patient records are highly confidential but extremely valuable for developing new models and statistical methods. The structure of patient record data is also closer to that of insurance data than most of the deep learning literature, focusing on unstructured data such as images. Images (and pixels) are continuous, whereas most of claimant characteristics are categorical variables, which adds complexity as one cannot interpolate between discrete classes to create fake records. Camino et al. (2018) adapted the medGAN and the Wassertein GAN with gradient penalty (WGAN-GP) of Gulrajani et al. (2017) for multi-categorical variables.

Deep learning has gained interest in recent actuarial research. Schelldorfer and Wüthrich (2019) analyze the French motor third party liability claims dataset studied in Section 5 with a generalized linear model embedded in a neural network. Wüthrich (2018b) use neural networks for chain-ladder reserving. To our knowledge however, only Kuo (2019) has used one type of GAN in actuarial applications so far.

In this paper, we introduce other GANs to the actuarial science literature and adapt the metrics to be appropriate for Poisson count data. Although there exists publicly available frequency datasets to develop and test pricing methods, they are really toy datasets compared to those that are kept confidential, due to the low number of policyholder or other covariates, like telematics or spatial information. We present and test three architectures. The first one, in Section 2, is based on the multi-categorical adaptation by Camino et al. (2018) of the WGAN-GP. Section 3 presents the conditional tabular GAN of Xu et al. (2019), applied to ratemaking data in Kuo (2019). The last model is what we call the MNCDP-GAN, detailed in Section 4, which is an adaptation of the differentially private GAN with autoencoder developed in Tantipongpipat et al. (2019). The MNCDP-GAN is the only one that incorporates differential privacy, which is the gold standard for guaranteeing that data can be shared without confidentiality issues. Section 5 shows a case study where the three architectures are tested on the French MTPL dataset, publicly available in the `R` package `CASdatasets` (Dutang and Charpentier, 2019). All of the code is available in the GitHub repository for this paper.[1] Section 6 concludes the paper and is followed by Appendix A and Appendix B which detail the setup and tuning of the multi-categorical and continuous WGAN-GP and the MNCDP-GAN, respectively.

2. Multi-categorical Wassertein GAN

Let us first introduce the general framework of generative adversarial networks. The training of a GAN is a game between two competing networks: the generator and the discriminator. The generator G is a neural net with parameter vector θ_g that takes in argument a vector of random noise Z with distribution F_z, and maps it to the space of the data we wish to model. Usually, the components of the vector Z are independent standard Gaussian random variables, and the dimension of Z is lower than the that of the data. The resulting $G(Z;\theta_g)$ is a fake data point, and its distribution is denoted by F_g.

The goal of the training procedure is therefore to find a good approximation F_g of the unknown distribution of a true data point X, denoted F_x. To achieve this goal, a competing network, the discriminator D with parameter vector θ_d, learns to determine whether a data point is real or fake. To this end, the parameters θ_d of D are trained to maximize the expected score of a real data point $\mathrm{E}_X\{D(X;\theta_d)\}$ and to minimize the expected score of a synthetic data point $\mathrm{E}_Z[D\{G(Z;\theta_g);\theta_d\}]$. To achieve the goal of generating realistic data points, the parameters θ_g of the generator are trained to maximize the discriminator's score on a *fake* data point $\mathrm{E}_Z[D\{G(Z;\theta_g);\theta_d\}]$. Combining the two problems together, the two networks aim to solve

$$\min_{\theta_g}\max_{\theta_d} \mathrm{E}_X[\log\{D(X;\theta_d)\}] + \mathrm{E}_Z\left[\log[1 - D\{G(Z;\theta_g);\theta_d\}]\right].$$

This optimization problem amounts to minimizing the Jensen-Shannon divergence between F_x and F_g. In practice, this leads to serious convergence issues, partly solved by training D and G in turn with minibatches.

[1] `https://github.com/brianmhartman/Anonymizing-Ratemaking-Datasets-using-GANs`

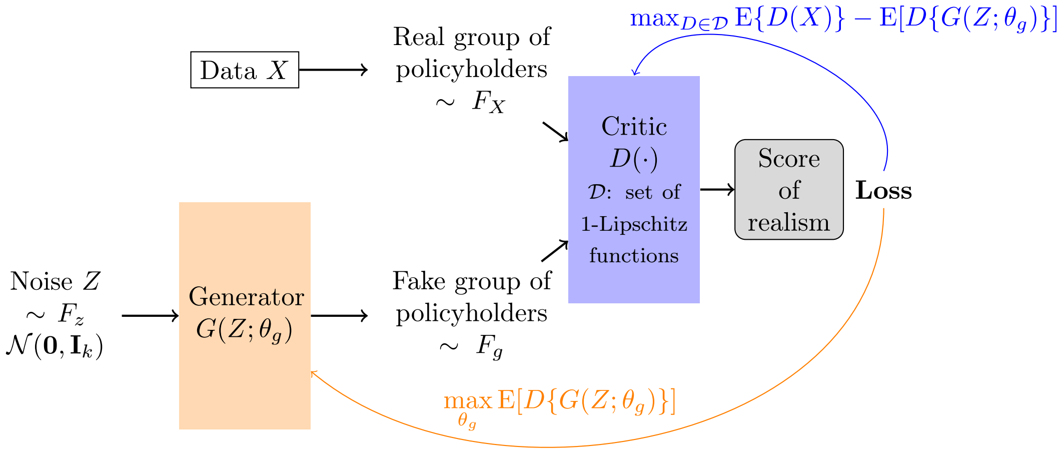
$\max_{D\in\mathcal{D}} \mathrm{E}\{D(X)\} - \mathrm{E}[D\{G(Z;\theta_g)\}]$
Data X
Real group of policyholders $\sim F_X$
Critic $D(\cdot)$
$\mathcal{D}$: set of 1-Lipschitz functions
Score of realism
Loss
Noise Z $\sim F_z$ $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$
Generator $G(Z;\theta_g)$
Fake group of policyholders $\sim F_g$
$\max_{\theta_g} \mathrm{E}[D\{G(Z;\theta_g)\}]$

Figure 1: WGAN Schema. The arrows represent the flow of the training process.

To solve some of the convergence issues, Arjovsky et al. (2017) advocate the use of the Wassertein-1 distance between F_x and F_g, that is, they consider the problem

$$\min_{\theta_g} \max_{D\in\mathcal{D}} \mathrm{E}_X\{D(X)\} - \mathrm{E}_Z\left[D\{G(Z;\theta_g)\}\right],$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. This change in the objective function leads to the Wassertein generative adversarial network, or WGAN. The discriminator in a WGAN is called the *critic*, as it is real valued rather than a binary classifier. The WGAN is depicted schematically in Figure 1 for policyholder claim data X. The black arrows represent the forward flow of information in the network, while the colored arrows represent the flow of the training process for the generator (orange) and the critic (blue).

Some tactics are needed to enforce the Lipschitz constraints on D. In this regard, the gradient penalty (GP) developed by Gulrajani et al. (2017) greatly improves the training of the WGAN. In their WGAN-GP, the authors take advantage of the fact that a differentiable Lipschitz function has gradients with norm at most 1 everywhere. A tuning parameter $\lambda > 0$ is introduced, and the objective of the WGAN-GP is

$$\min_{\theta_g} \max_{\theta_d} \mathrm{E}_X\{D(X;\theta_d)\} - \mathrm{E}_Z\left[D\{G(Z;\theta_g);\theta_d\}\right] + \lambda \mathrm{E}_{\hat{X}}[\{||\nabla_{\hat{x}} D(\hat{X};\theta_d)||_2 - 1\}^2],$$

where $\hat{X} \overset{d}{=} UX+(1-U)G(Z;\theta_g)$, and U is uniformly distributed on the interval $(0,1)$, so that the distribution $F_{\hat{x}}$ of $\hat{X}$ is obtained by sampling uniformly along lines between pairs of points sampled from F_x and F_g. For details on the motivation, the reader is referred to Gulrajani et al. (2017).

In practice, if $m \in \mathbb{N}$ is the size of the minibatch with observations $x_1, \ldots, x_m$, random noise vectors $z_1, \ldots, z_m$ and independent uniform samples $u_1, \ldots, u_m$, then we let $\hat{x}_i = u_i x_i + (1-u_i)G(z_i;\theta_g)$ and the discriminator loss is approximated by

$$\mathcal{L}_d = \frac{1}{m}\sum_{i=1}^{m} -D(x_i, \theta_d) + D\{G(z_i;\theta_g);\theta_d\} + \lambda\{||\nabla_{\hat{x}_i} D(\hat{x}_i;\theta_d)||_2 - 1\}^2$$

while the generator loss is simply

$$\mathcal{L}_d = \frac{1}{m}\sum_{i=1}^{m} -D\{G(z_i;\theta_g);\theta_d\}.$$

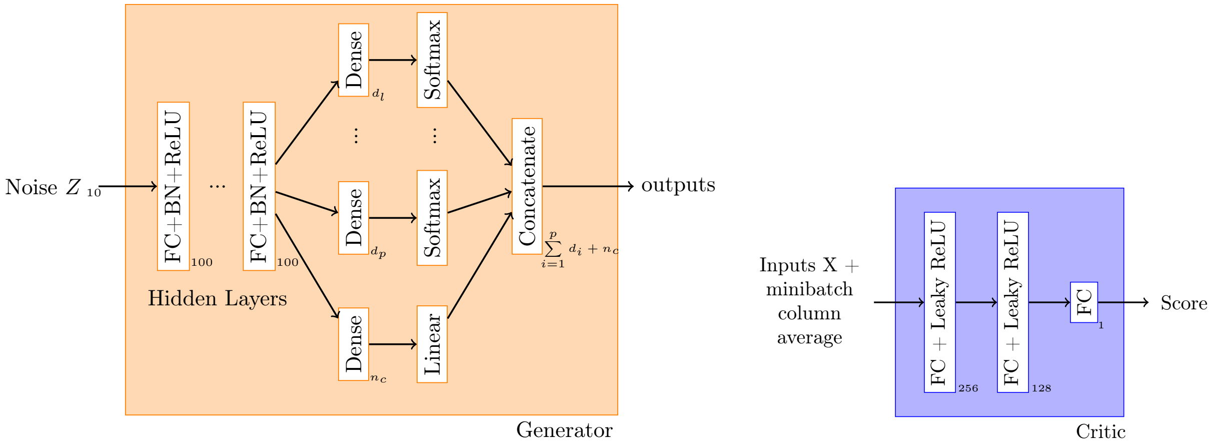

Figure 2: Architecture inside the generator (orange) and the critic (blue) for our multi-categorical and continuous WGAN. The dimensions $d_1, \ldots, d_p$ represent the number of levels in categorical variables $1, \ldots, p$. FC stands for fully connected and BN stands for Batch normalization.

Note that higher values of the critic D indicate fake samples.

The WGAN and the WGAN-GP were developed in the context of image generation tasks. In the current application however, we wish to synthesize tabular insurance data, in which some variables are categorical with multiple levels. An application closer to ours was considered by Camino et al. (2018), where the target data contains many multi-categorical variables. Camino et al. (2018) modified the generator of the WGAN-GP so that, after the model output, there is a dense layer in parallel for each categorical variable followed by a softmax activation function. Then, the results are concatenated to yield the final generator output.

As in Camino et al. (2018), our generator's architecture has one dense layer with dimension matching the number of levels for each multi-categorical variable. We also add one dense layer with linear activation and dimension n_c which is equal to the number of continuous variables. The architecture of the generator and the critic in our multi-categorical and continuous WGAN-GP, or MC-WGAN-GP, is depicted in Figure 2. Further details about hyperparameter optimization are available in Appendix A.

3. CTGAN

Another possible path to simulating insurance claim data is called a conditional tabular GAN or CTGAN (Xu et al., 2019). This method was applied to ratemaking data in Kuo (2019). Additionally, Kuo developed an R wrapper for this software to make it easily accessible to insurance practitioners more familiar with R than Python. Starting from his code, we slightly adjusted the preamble to improve the application consistency on our machines and we adjusted the preprocessing slightly, but other than that the overall code remained the same. Our version of the code is available in the GitHub repository for this paper.

The CTGAN simulates records one by one. It first randomly selects one of the variables (say fuel type, diesel or gasoline). Then, it randomly selects a value for that variable (say diesel). Following Kuo (2019), we use the true data frequency to sample the value rather than the log-frequency as suggested in Xu et al. (2019). Given that value for that variable, the algorithm finds a matching row from the training data (in this example, it randomly selects a true observation with a diesel-powered car). It also generates the rest of the variables conditioning on it being diesel-powered. The generated and true rows are sent to the critic which gives a score. Figure 3 summarizes the CTGAN procedure.

Figure 4 zooms inside the architecture of the generator and the critic. Both the critic (blue) and the generator (orange) use two fully-connected layers to attempt to capture all relationships between the columns. The generator uses skip connections and allows only for categorical variables; see Xu et al. (2019) for the non-trivial extension with continuous variables. An additional sophistication of the CTGAN is the use of

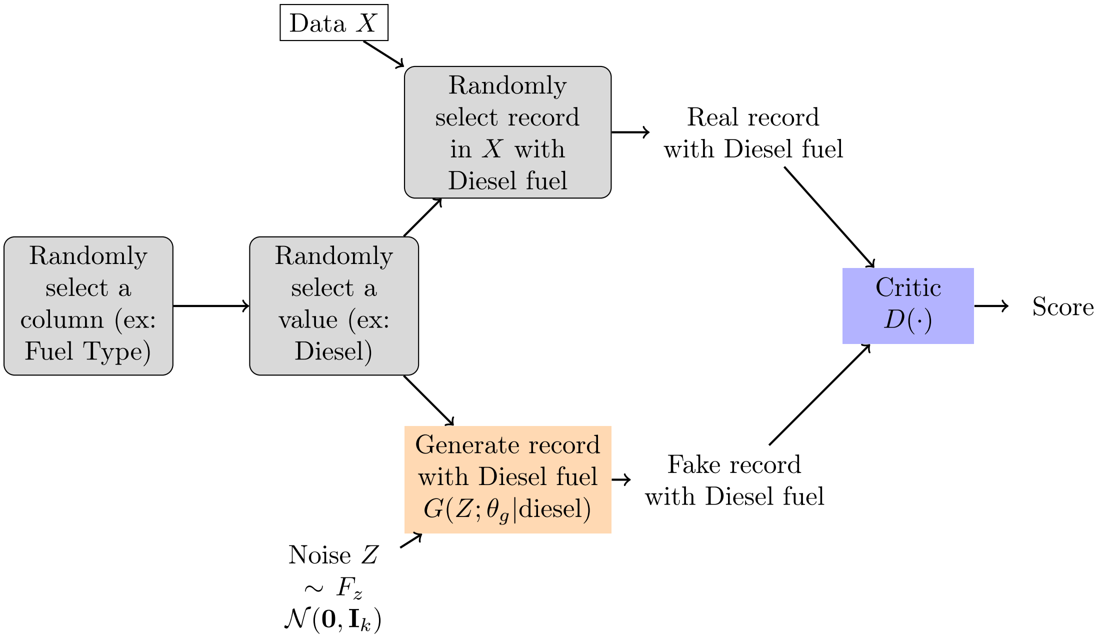
Data X
Randomly select record in X with Diesel fuel
Real record with Diesel fuel
Randomly select a column (ex: Fuel Type)
Randomly select a value (ex: Diesel)
Critic $D(\cdot)$
Score
Generate record with Diesel fuel $G(Z; \theta_g|\text{diesel})$
Fake record with Diesel fuel
Noise Z $\sim F_z$ $\mathcal{N}(\mathbf{0}, \mathbf{I}_k)$

Figure 3: CTGAN Schema. The model flow is illustrated for the case when the selected column is the Fuel Type and the selected value for that column is Diesel.

the PacGAN framework (Lin et al., 2018) in the discriminator, where ten samples are provided in each pac to prevent the mode collapse issue.

Like the previously discussed MC-WGAN-GP, the CTGAN does not incorporate privacy protections, though that could be possible to develop, as hinted in Kuo (2019). This implementation also only simulates entirely discrete data, which is a disadvantage when working with continuous information such as exposure.

4. MNCDP-GAN

The Mixed Numerical and Categorical Differentially Private GAN (MNCDP-GAN) tries to solve the drawbacks of the two other GANs. It includes an autoencoder and a WGAN. The main advantage of this architecture, introduced in Tantipongpipat et al. (2019), is that the generator works in a latent space of encoded variables, which can be easier to model adequately than the original structured data. The training can be done in a differentially private (DP) manner, allowing a DP guarantee on the generated dataset.

As depicted in Figure 5, the original data is first pre-processed (one-hot encodings for categorical variables and either binning or min-max standardization for continuous variables), resulting in vectors defined in $[0,1]^n$ that are fed into an encoder shrinking the dimension to $d < n$, a hyper-parameter. Then, a decoder takes in entry the encoded variable in the latent space $\mathbb{R}^d$ and outputs data in the format $[0,1]^n$, which is subsequently post-processed to lead to data in the original format. This architecture is called an autoencoder and is used in many neural network applications. In our context, the autoencoder creates the latent space in dimension d, which is easier to learn for the generator as it has less structure than the original data space. The generator takes in random noise and outputs a vector in the latent space, which can then be decoded by the decoder to produce a synthesized record. The critic is trained with the Wassertein loss and compares the generated data before postprocessing with the preprocessed original data.

In Figure 5, the autoencoder flow and training are depicted in red, the flow of the data in the autoencoder and the critic is indicated in green, and the flow of the generated data through the decoder and the critic is highlighted in orange font. It is assumed, and reasonable, that the postprocessing step can be done using public knowledge and does not affect the DP quality of the model. The DP training is done by injecting noise in the decoder and the critic, for more details, refer to Tantipongpipat et al. (2019).

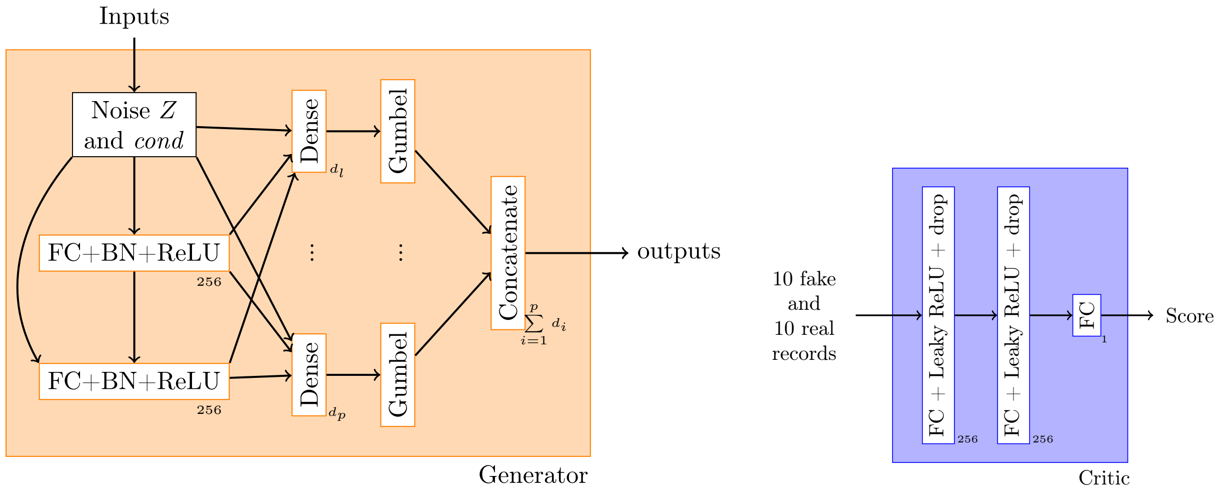

Figure 4: Architecture inside the generator (orange) and the critic (blue) for the CTGAN. The dimensions $d_1, \dots, d_p$ represent the number of levels in categorical variables $1, \dots, p$. All variables are assumed categorical. The input of the generator is Gaussian random noise and the condition *cond* of the feature value that was randomly selected (see Figure 3). FC stands for fully connected, BN for Batch normalization, Gumbel for the Gumbel softmax activation, and drop for dropout.

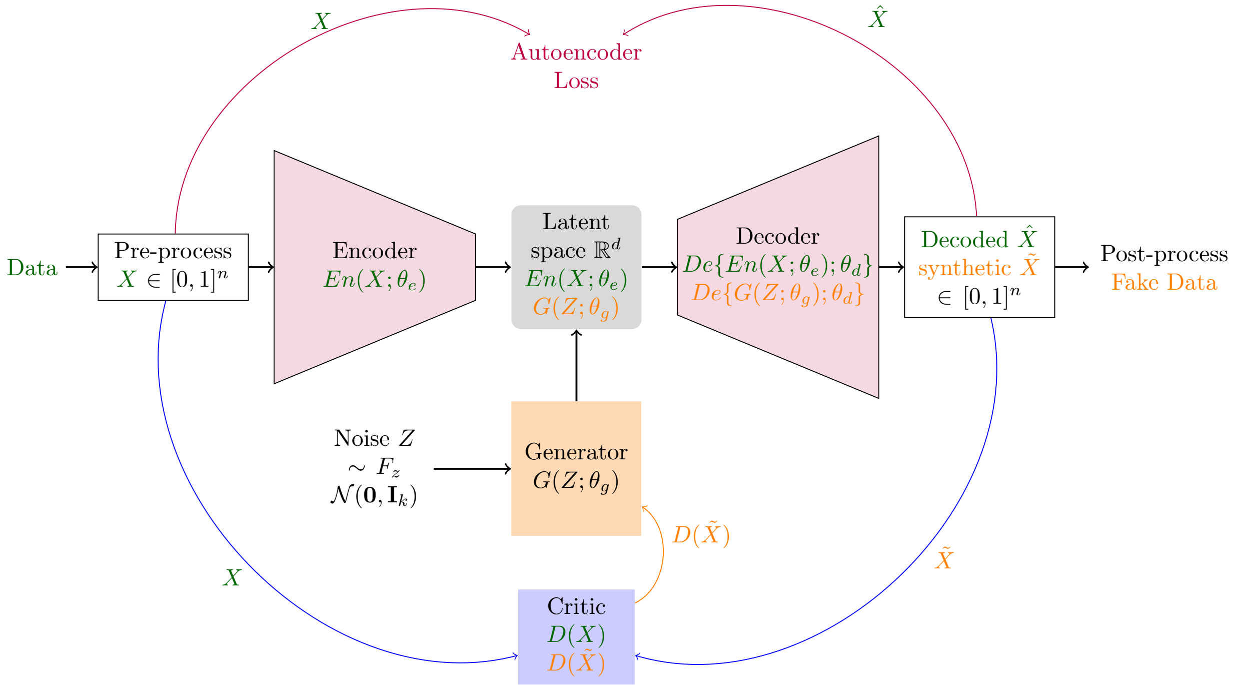

Figure 5: MNCDP-GAN Schema. The orange relates to the generated data while the green relates to the original data. The colored arrows represent the flow into the loss for the training of each network; autoencoder in red, generator in orange and discriminator in blue. For DP training, noise is added in training the decoder and the critic.

The level of differential privacy that is achieved by the model (including both the autoencoder and the GAN) is quantified by the value $\epsilon > 0$. This value relates to how different an analysis may be if one datapoint is added or removed. If the dataset X' is identical to X except for one added data point, then an (ϵ, δ) differentially private analysis $\mathcal{M}$ satisfies

$$\Pr\{\mathcal{M}(X) \in S\} \leq e^{\epsilon} \Pr\{\mathcal{M}(X') \in S\} + \delta$$

for $\delta > 0$, any such X, X' and $S \subseteq \mathit{Range}(\mathcal{M})$ (see, e.g., Dwork and Roth, 2014). A smaller ϵ represents stronger privacy guarantees, but as more noise is added to the training, it also comes with decreased performance. The values of ϵ and δ for our procedure are obtained through a *privacy accountant* as explained in Tantipongpipat et al. (2019). Further details on our implementation are available in Appendix B.

5. Case Study

To show the value of the three approaches in a reproducible manner and to compare their effectiveness in producing synthetic data, we use a well-known publicly available dataset for the case study. It contains a set of 412,748 French motor third-party liability policies observed in a single year (Dutang and Charpentier, 2019). There are eight explanatory variables in the data:

- Exposure: the number of car-years on the policy, bounded between 0 and 1 (we removed the few records with exposure greater than one)
- Power: an ordered categorical variable which describes the power of the vehicle
- CarAge: the vehicle age in years
- DriverAge: the primary driver age in years
- Brand: the vehicle brand divided in the following groups: A – Renault, Nissan, and Citroen, B – Volkswagen, Audi, Skoda, and Seat, C – Opel, General Motors, and Ford, D – Fiat, E – Mercedes, Chrysler, and BMW, F– Japanese (except Nissan) and Korean, G – other.
- Gas: diesel or regular
- Region: the policy region in France
- Density: number of inhabitants per km^2 in the home city of the driver

Brand, Gas, Power, and Region are all categorical variables and the other four are numeric (continuous or discrete). Since the CTGAN only accommodates categorical variables, the continuous variables are binned for that setup. For the MNCDP models we will show four different DP levels:

- $\epsilon = \infty$ is labeled on the plots as "MNCDPInfty": no differential privacy
- $\epsilon = 100,000$ labeled as "MNCDP100k"
- $\epsilon = 10,000$ labeled as "MNCDP10k"
- $\epsilon = 5$ is labeled as "MNCDP5": strong differential privacy

We simulate a dataset of the same size as the original dataset using each of the GANs, and we compare the univariate distributions in the generated samples with the univariate distributions in the real data. If the methods are able to faithfully reproduce the original data, then we expect the distributions to be similar.

We first compare the results for the categorical variables. Figure 6 shows the observations in each category in the real and generated datasets for brand, gas type, power and region for the real data, the MC-WGAN-GP, the CTGAN and the MNCDP-GAN without DP. We see that from the univariate perspective, the three models all replicate the real data reasonably well. In particular, the MC-WGAN-GP (blue) reproduces closely the univariate distributions in the real data (red) for these four categorical variables.

Figure 7 shows the same information as Figure 6 but for the MNCDP model with varying levels of differential privacy. It becomes readily apparent that the synthesized data gets dramatically worse as the level of differential privacy increases. Again, the model with $\epsilon = \infty$, no differential privacy, follows the data rather well. As ϵ decreases to 100,000, the model still maps relatively well to the real data. But, $\epsilon = 10,000$ and especially $\epsilon = 5$, are not close to the real data. The noise added to the process in both cases completely obscures the original signal. This will be a consistent result throughout our case study.

Shown another way, Figure 8 plots the frequency for each category in the real data on the x-axis against the frequency in the synthesized data on the y-axis for each of the GAN considered, color-coded by feature. The line $y = x$ is also plotted. The MC-WGAN-GP dataset seems to match the real frequencies the best, followed by the CTGAN, MNCDPInfty, and MNCDP100k models (which are relatively similar). As noticed above, the MNCDP10k and MNCDP5 models are drastically different from the original data.

For the numeric variables CarAge, Density, DriverAge and Exposure, Figure 9 shows the distributions of the real data on the top row and compares it to the distributions of data generated by the models. One of the most difficult aspects of insurance data is the exposure variable. It has a large proportion which are exactly 1. After accounting for those, the rest of the data tends to be either close to 0 or close to monthly intervals (1/12, 2/12, 3/12, etc.). Both the CTGAN and the MC-WGAN-GP do well synthesizing the correct number of 1 values, but when looking at the rest of the distribution, the MC-WGAN-GP is too bumpy and the CTGAN might be too smooth. For the Density random variable, CTGAN is the best. Both DriverAge and CarAge are matched by the three methods rather well.

It is important to correctly model the univariate characteristics, but it is even more important to correctly model the multivariate relationships. This is especially true with the relationship between claim counts and the various explanatory variables. Figure 10 compares the probability of a claim in each categorical group. The real probability of a claim in on the x-axis while the synthesized probability is on the y-axis. The line $y = x$ is also plotted to show the ideal goal. Also, the size of the marks shows the proportion of the synthesized data in each group. By this metric, the MC-WGAN-GP performs the best, followed closely by the CTGAN. The MNCDP-GAN does not do well, even without differential privacy.

Our last test examines how consistent are models fitted on the original and the synthesized data. We split the real data into two parts, a 70% training set and a 30% test set. We fit a Poisson GLM on the training set predicting the number of claims using all eight of the explanatory variables. We then fit the same model on a 70% sample from each of the synthesized datasets. The sampling and model fitting are performed 5,000 times to understand the sampling variability and to obtain more consistent estimates. In Figure 11, we compare the average estimated regression coefficients for each of the three models. We found that the CTGAN coefficients are all close to the real coefficients. The coefficients estimated with the MC-WGAN-GP data are similarly close with the exception of a single region coefficient. The results achieved using the MNCDP synthesized data are again the worst.

With each fitted model, we then predicted the claim counts for the 30% real test data and compared the predictions from the models fit on the synthesized data to the predictions from the models fit on the real data. Table 1 shows the median absolute error (MAE) and mean squared error (MSE) between the predictions, with 95% bootstrapped confidence intervals. The MC-WGAN-GP significantly outperforms the other two models on both metrics. The CTGAN performs next best and the worst performance is again attributed to the MNCDP-GAN.

Table 1: Poisson regression prediction errors (and bootstrapped 95% intervals) for the three main GANs.

Model	MAE (×1000)	MSE (×1000)
MC-WGAN-GP	5.0 (4.68, 5.44)	0.08 (0.06, 0.10)
CTGAN	10.8 (10.28, 11.32)	0.38 (0.36, 0.40)
MNCDPInfty	32.8 (32.38, 33.22)	3.36 (3.20, 3.52)

6. Conclusion

In this paper, we presented, implemented and compared three different methods to synthesize insurance data. All of the methods are based on generative adversarial networks. All three methods have advantages and disadvantages. The MC-WGAN-GP method synthesized the most realistic data. It generated data which was very similar (accounting for both univariate and multivariate relationships) to the real data. The CTGAN method was the easiest to use, especially for someone more familiar with R than Python. The CTGAN synthesized data was almost as good as the MC-WGAN-GP data. The main drawbacks of MC-WGAN-GP and CTGAN is that there are no privacy guarantees, meaning that some records in the generated data could still contain confidential information. The MNCDP-GAN incorporates differential privacy, but the synthesized data (even without differential privacy) was not as good as with the other two methods.

Future work can start from any of the three models and attempt to add the advantages of the other two. Either add differential privacy and ease of use to the MC-WGAN-GP, improved synthesis and differential privacy to the CTGAN, or improved synthesis and ease of use to the MNCDP-GAN. In any case, GANs are a promising tool for synthesizing and protecting private and important data.

7. Acknowledgments

This work was supported by a Casualty Actuarial Society (CAS) Individual Grant and by M.-P. Côté's Chair in Educational Leadership in Big Data Analytics for Actuarial Sciences — Intact. The authors thank the members of the CAS project oversight committee, Syed Danish Ali, Morgan Bugbee, Marco De Virgilis, and Greg Frankowiak, for useful feedback throughout the project. The project would not have been possible without the support provided by Calcul Québec (www.calculquebec.ca), Compute Canada (www.computecanada.ca), and the statistics department computing cluster at BYU.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., Zhang, L., 2016. Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 308–318.
URL `arXiv1607.00133v2`

Antonio, K., Plat, R., 2014. Micro-level stochastic loss reserving for general insurance. Scandinavian Actuarial Journal 2014 (7), 649–669.

Arjovsky, M., Chintala, S., Bottou, L., 06–11 Aug 2017. Wasserstein generative adversarial networks. In: Precup, D., Teh, Y. W. (Eds.), Proceedings of the 34th International Conference on Machine Learning. Vol. 70 of Proceedings of Machine Learning Research. PMLR, International Convention Centre, Sydney, Australia, pp. 214–223.
URL `http://proceedings.mlr.press/v70/arjovsky17a.html`

Camino, R., Hammerschmidt, C., State, R., 2018. Generating multi-categorical samples with generative adversarial networks. arXiv preprint arXiv:1807.01202.

Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F., Sun, J., 18–19 Aug 2017. Generating multi-label discrete patient records using generative adversarial networks. In: Doshi-Velez, F., Fackler, J., Kale, D., Ranganath, R., Wallace, B., Wiens, J. (Eds.), Proceedings of the 2nd Machine Learning for Healthcare Conference. Vol. 68 of Proceedings of Machine Learning Research. PMLR, Boston, Massachusetts, pp. 286–305.

Dutang, C., Charpentier, A., 2019. CASdatasets: Insurance datasets. R package version 1.0-10.

Dwork, C., Roth, A., 2014. The algorithmic foundations of differential privacy. Foundations and Trends in Theoretical Computer Science 9 (3-4), 211–407.

Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., Greenspan, H., 2018. Synthetic data augmentation using GAN for improved liver lesion classification. In: 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018). IEEE, pp. 289–293.

Gabrielli, A., Wüthrich, M. V., 2018. An individual claims history simulation machine. Risks 6 (2), 29.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., Weinberger, K. Q. (Eds.), Advances in neural information processing systems 27. Curran Associates, Inc., pp. 2672–2680.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A. C., 2017. Improved training of Wasserstein GANs. In: Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 30. Curran Associates, Inc., pp. 5767–5777. URL http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf

Kuo, K., 2019. Generative synthesis of insurance datasets. arXiv preprint arXiv:1912.02423.

Lin, Z., Khetan, A., Fanti, G., Oh, S., 2018. PacGAN: The power of two samples in generative adversarial networks. In: Advances in neural information processing systems. pp. 1498–1507.

Papernot, N., Abadi, M., Erlingsson, Ú., Goodfellow, I., Talwar, K., 2017. Semi-supervised knowledge transfer for deep learning from private training data. In: Proceedings of the International Conference on Learning Representations.
URL https://arxiv.org/abs/1610.05755

Pigeon, M., Antonio, K., Denuit, M., 2013. Individual loss reserving with the multivariate skew normal framework. ASTIN Bulletin: The Journal of the IAA 43 (3), 399–428.

Pigeon, M., Antonio, K., Denuit, M., 2014. Individual loss reserving using paid–incurred data. Insurance: Mathematics and Economics 58, 121–131.

Schelldorfer, J., Wüthrich, M. V., 2019. Nesting classical actuarial models into neural networks.
URL https://ssrn.com/abstract=3320525

Tantipongpipat, U., Waites, C., Boob, D., Siva, A. A., Cummings, R., 2019. Differentially private mixed-type data generation for unsupervised learning. arXiv preprint arXiv:1912.03250.

Wüthrich, M. V., 2018a. Machine learning in individual claims reserving. Scandinavian Actuarial Journal 2018 (6), 465–480.

Wüthrich, M. V., 2018b. Neural networks applied to chain-ladder reserving. European Actuarial Journal 8 (2), 407–436.

Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K., 2019. Modeling tabular data using conditional GAN. In: Advances in Neural Information Processing Systems. pp. 7333–7343.

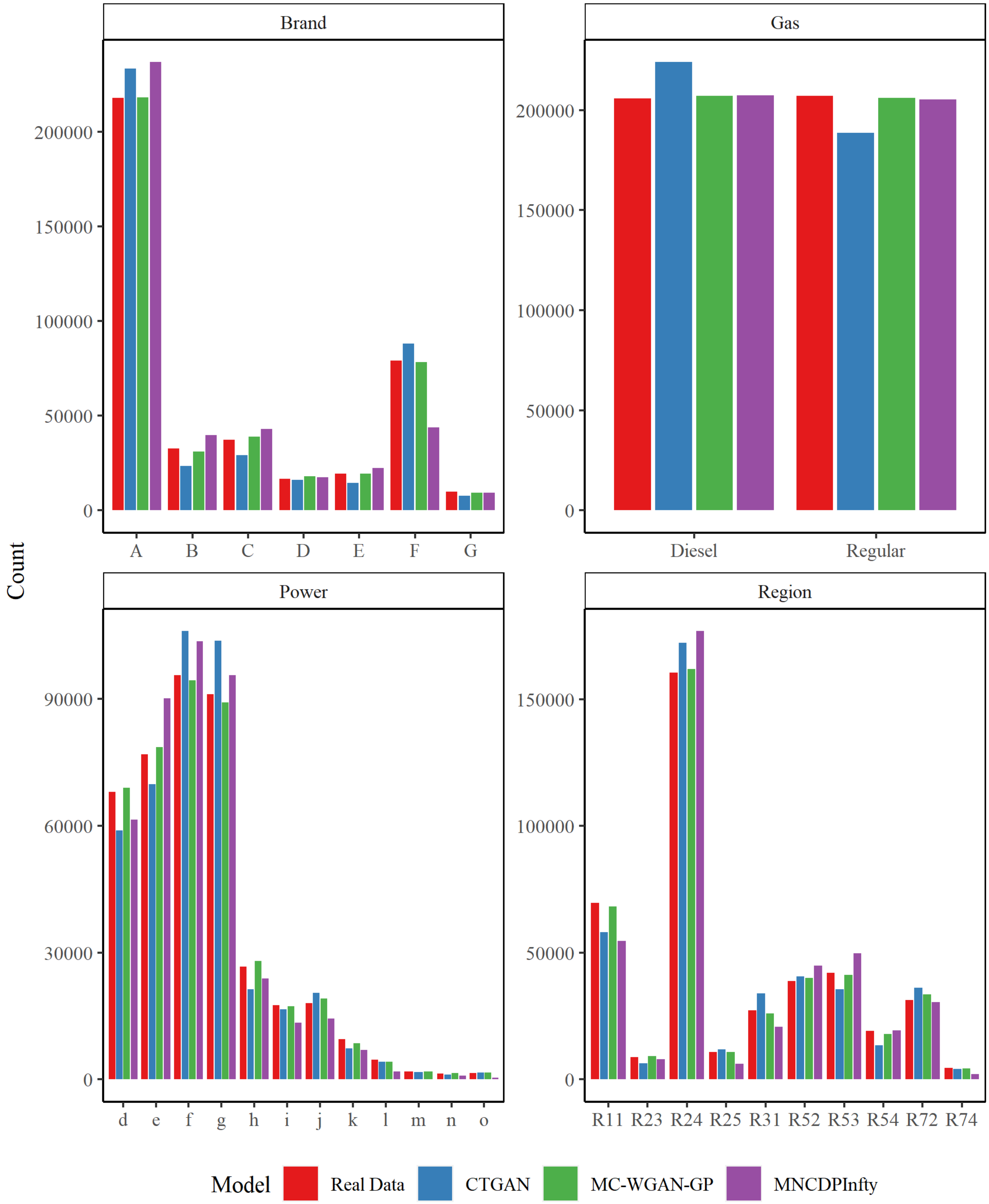

Figure 6: Comparison of univariate categorical variable distributions.

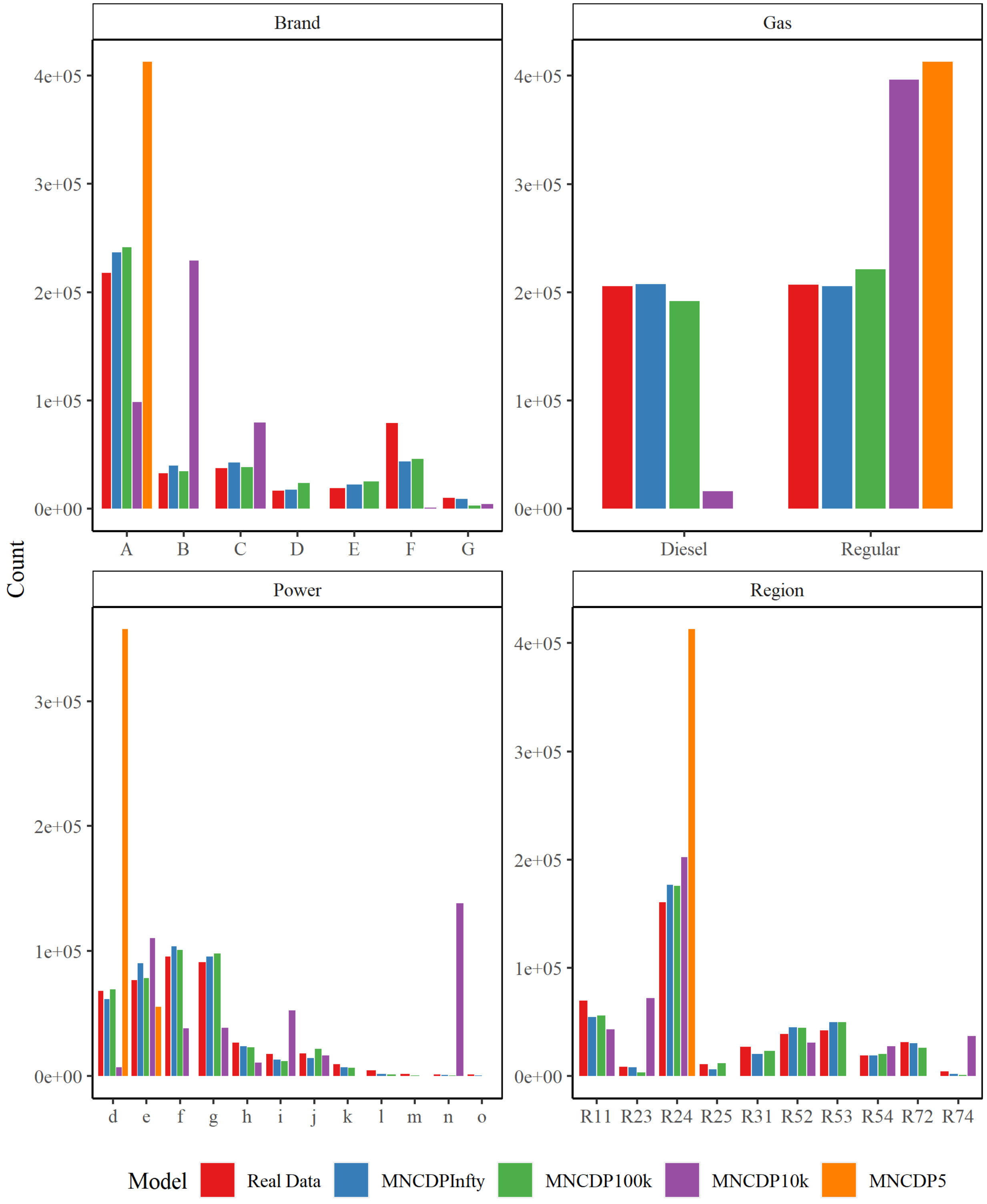

Figure 7: Comparison of univariate categorical variable distributions for MNCDP-GAN models with $\epsilon \in \{5, 10000, 100000, \infty\}$.

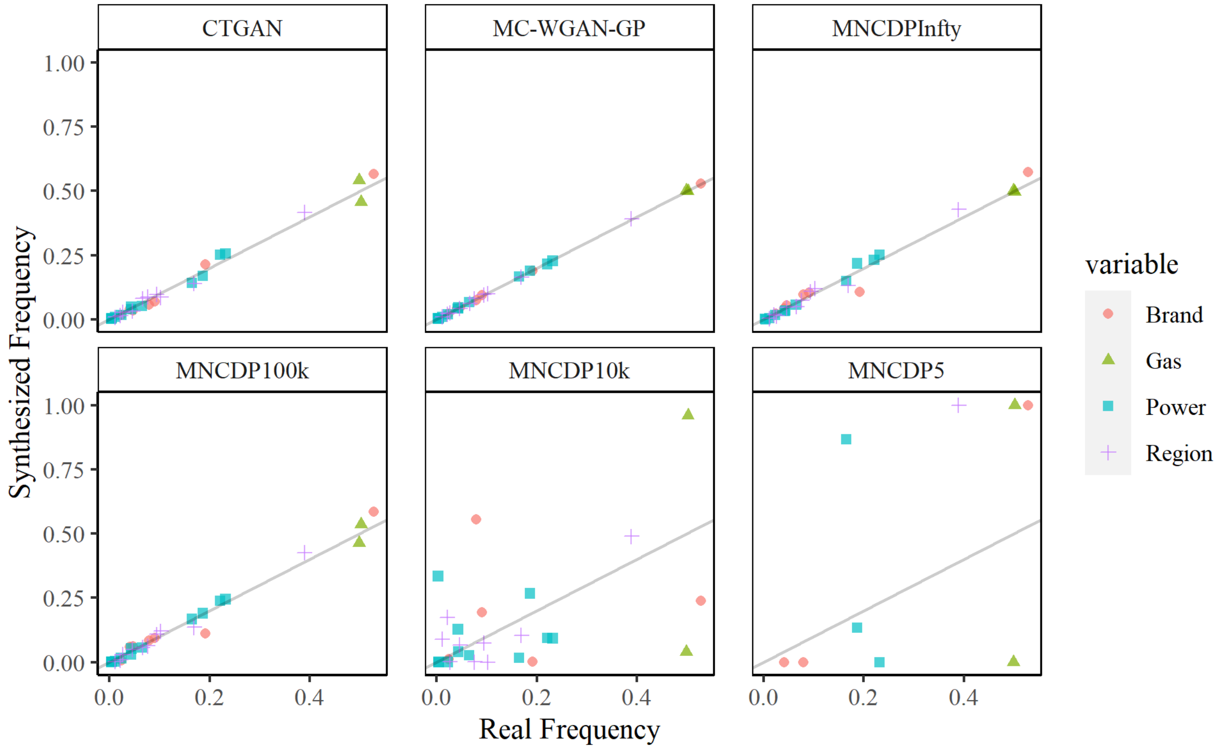

Figure 8: Comparison of the synthesized and real group frequencies for each class of the four categorical variables.

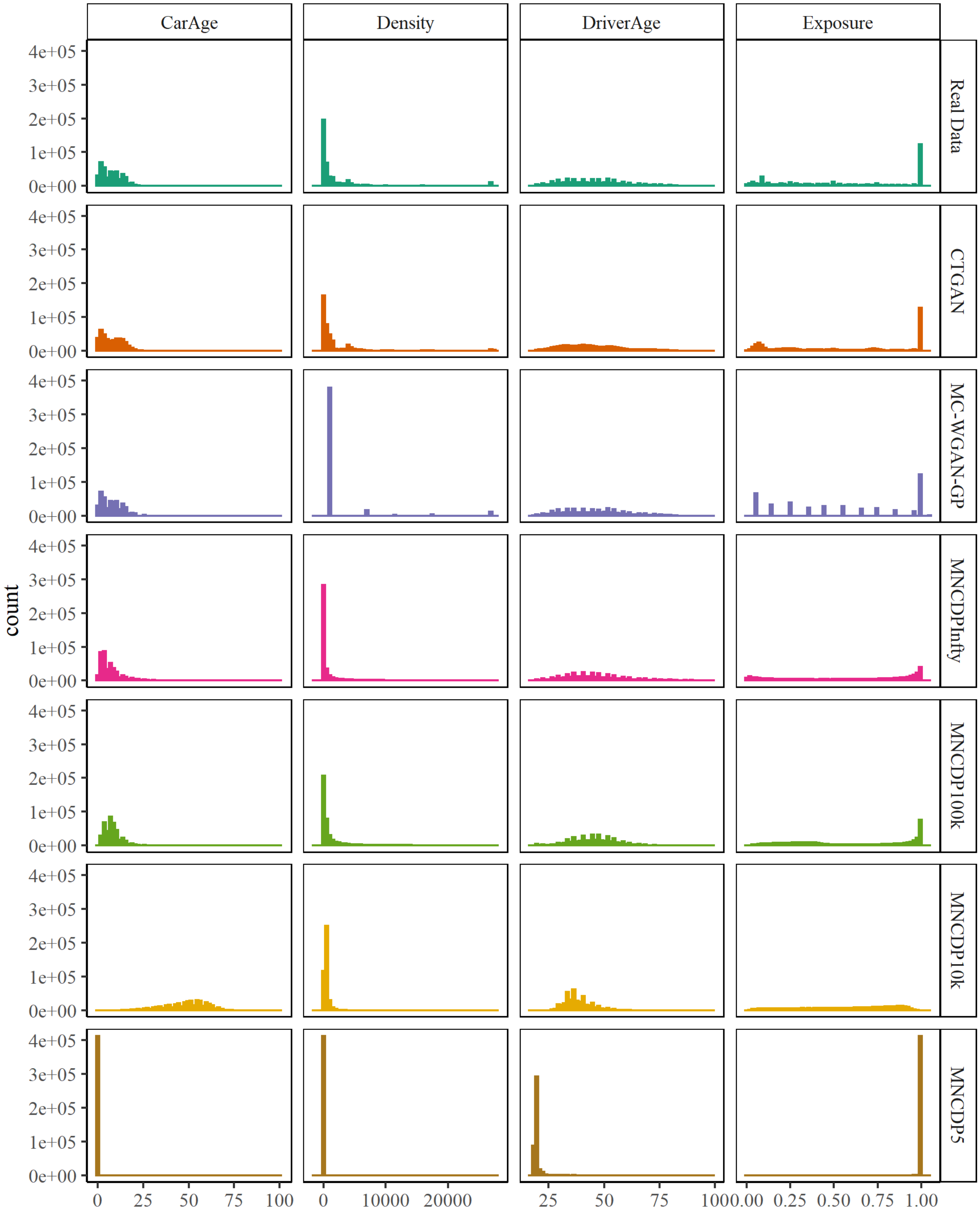

Figure 9: Comparison of univariate numerical variable distributions.

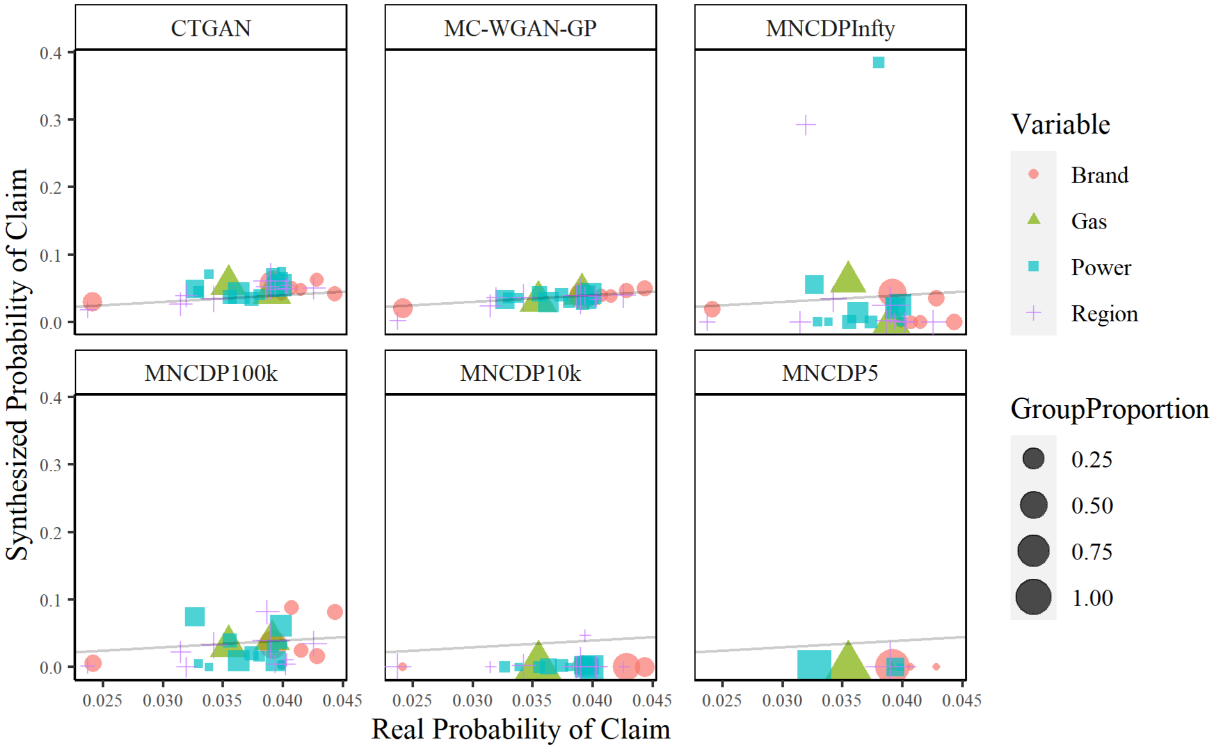

Figure 10: Comparison of the synthesized and real empirical probability of a claim for each group.

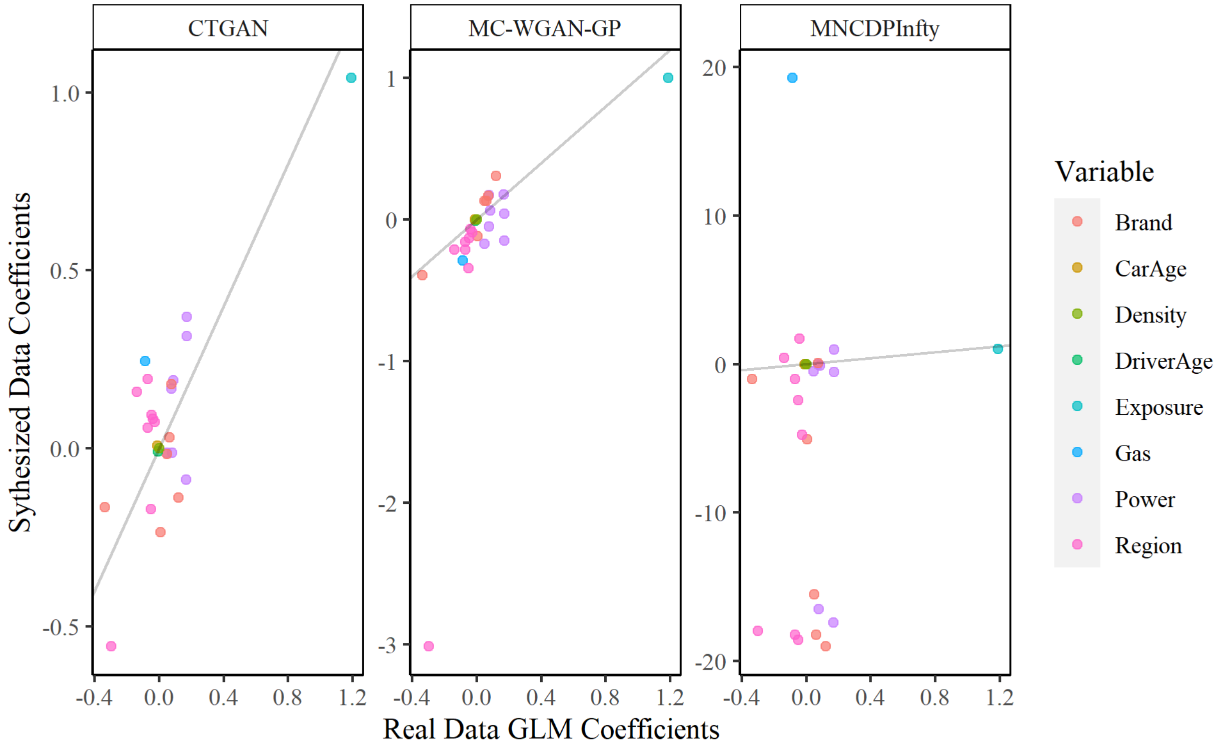

Figure 11: Comparison of the Poisson regression coefficients for the synthesized and real data.

Appendix A. MC-WGAN-GP Hyperparameter Tuning

In the MC-WGAN-GP model, we tuned the following hyperparameters.

- Loss Penalty – loss_penalty
- Generator Batch Norm Decay – gen_bn_decay
- Discriminator Batch Norm Decay – disc_bn_decay
- Generator L2 Regularization – gen_L2_reg
- Discriminator L2 Regularization – disc_L2_reg
- Learning Rate – learning_rate

We used a random search to explore the settings and decided on the values in Table A.2.

Table A.2: Hyperparameter settings for the MC-WGAN-GP.

Hyperparameter	Explored Values	Chosen Value
loss_penalty	1, 5, 10, 20, 50	10
gen_bn_decay	0, 0.10, 0.25, 0.45, 0.50, 0.90	0.90
gen_L2_reg	0, 0.00001, 0.0001, 0.001, 0.01	0
disc_L2_reg	0, 0.00001, 0.0001, 0.001, 0.01	0
learning_rate	0.001, 0.005, 0.01	0.01

Appendix B. MNCDP-GAN Methodology

In this appendix, we give details on the preprocessing of the variables in the French Motor Third-Party Liability frequency dataset for the MNCDP-GAN in Appendix B.1. We then explain the training procedure in Appendix B.2, the hyperparameter optimization in Appendix B.3 and the final setting in Appendix B.4.

Appendix B.1. Preprocessing

Because the Exposure variable should be capped at one, all 421 samples whose Exposure was above this threshold were removed. As for the target ClaimNb, it was converted to a categorical variable since there are only five possible values (0 to 4) and it is highly skewed towards zero. The adjusted dataset, which was the one used to train the models, contains 412,748 samples explained by four numerical (DriverAge, CarAge, Exposure and Density) and five categorical (ClaimNb, Power, Brand, Gas and Region) variables.

For the MNCDP-GAN experiments, multiple configurations differing on the types of those variables were tested. The first one, called "baseline", uses this data as is. In the "all-cat" configuration, both Exposure and Density were binned into categorical variables, as was also done for the MC-WGAN-GP. In the "bin" configuration, the four numeric variables were binned in order to be treated as categorical. Expert insight was required to determine the binning size for each feature independently.

In the case of DriverAge, ten bins of increasing size (more bins at younger ages) were made to approximate a normal distribution. For CarAge, a single bin was used for the value 0 while the rest was split using ten quantiles. The resulting distribution is thus closer to the uniform distribution. For Exposure, 12 bins were chosen to reflect the 12 months of a year with the exception of the value one which has a bin of its own. The resulting distribution is skewed towards that last bin. Finally, for Density, the binning was applied on its natural logarithm and based on the deciles, so that the resulting distribution is almost uniform.

Appendix B.2. Training

For each configuration of the MNCDP-GAN, the autoencoder (AE) and the GAN were trained independently one after the other, since the GAN requires the decoder. The training of the AE was done over 20,000 iterations, which was determined to be sufficient for convergence. Before feeding the preprocessed data to the network, the numerical features were normalized using min/max normalization while the categorical variables were encoded using one-hot vectors. Unlike the more common way to encode N categories in $N-1$ dimensions, it was decided to use one dimension per category instead. Otherwise, the GAN would never generate the category not having a dimension of its own. The AE uses the binary cross entropy loss.

The training of the GAN was done over 2M iterations. This value was determined empirically based on the results obtained. However, because of the known difficulty to train GANs (no stability guarantees), it could be adjusted depending on the configuration and the hyperparameters. The generator and discriminator use the zero-sum objective function as proposed by Arjovsky et al. (2017) for their learning. As recommended in this same paper, the discriminator was updated more often than the generator in order to train until optimality. Using a linear activation as the output layer of the discriminator and the RMSProp optimizer for the GAN are also recommendations from these authors that were applied.

For both the AE and the GAN, the preprocessed dataset was split 2/3 for training and 1/3 for validation. The basic architecture of the networks (the number and sequence of layers) was not changed from Tantipongpipat et al. (2019). Unless stated otherwise, the activation functions used were always LeakyReLU with a negative slope of 0.2. Because they depend on the input size and some hyperparameters (such as the latent dimensions), the sizes of the layers vary from configuration to configuration. The design choices of using the ADAM optimizer with gradient penalty for the AE, choosing the absolute bounds to clip the values of the WGAN gradients and choosing layer normalization over batch normalization for the generator follow the recommendations of Gulrajani et al. (2017).

The algorithm of the differentially private stochastic gradient descent (DP-SGD) can be found in Tantipongpipat et al. (2019). For the differential privacy aspect, computing the L2 clipping norms of the gradients for the decoder and the discriminator was done as recommended by Abadi et al. (2016). Finally, before training the models used to obtain results, the hyperparameters were tuned using a random search.

Appendix B.3. Hyperparameter optimization

As for the training, the tuning of the hyperparameters of the AE and the GAN was done in two stages. In a random search, each combination of hyperparameters tested is selected randomly from the chosen grid. The number of search iterations is set based on a time/resources compromise. The strategy was to run multiple searches instead of a single big one. Each time, the search space was narrowed for more fine-tuning. In all cases, the tuning was done in a non private way ($\epsilon = \infty$) in order to reduce computation time.

For the AE, the hyperparameters tested were the minibatch size, compression dimension, learning rate, β_1 and β_2 parameters of the ADAM optimizer, and the L2 penalty of the weight decay for the optimizer. In first experiments, these last three hyperparameters did not affect the training significantly so it was decided to leave them at their usual default values (0.9, 0.999 and 0, respectively). To evaluate the performance of each combination, the validation and training losses were saved and sorted. The combination with the lowest final validation loss was chosen. A regular training was then done to confirm that it was not overfitting.

For the GAN, the hyperparameters tested were the minibatch size, the latent dimension of the generator, the learning rate, the number of iterations of the discriminator before updating the generator once, the L2 penalty of the weight decay of the optimizer and the α smoothing constant of the RMSProp optimizer. Once again, these last two hyperparameters of the optimizer did not affect the results significantly so they were left at 0 and 0.99 respectively. The evaluation of the performance of the GAN was not as straightforward as for the AE. Both the losses of the discriminator and the generator were plotted. Over time, it was found that the desired loss curve of the discriminator was one which dropped rapidly near 0 and that then converged to that value over training iterations. For the generator, a loss oscillating rather slowly around 0 (going positive for many thousands of iterations then going negative and so on) appeared a good indicator of performance. Fortunately, these tendencies could be spotted for training of only a few tens of thousands of iterations. Hence, to reduce computation time, the number of iterations for each combination was limited to 100,000.

Once a couple of potentially good combinations were identified in this manner, a full training over 2M iterations was done for each one. The generated samples of these trained models were then evaluated in respect to the univariate distributions of each variable (vs the real distributions). The predictions on the target ClaimNb of a random forest regressor and of a random forest classifier were also compared between the generated and the real samples. The combination giving the best overall results was kept. Because of the lack of stability of the GANs (even for the same hyperparameters and configuration), this best combination of hyperparameters was trained at least two more times with different seeds for the random number generator. The model of whichever run gave the best results was saved and used for the final results.

For both the autoencoder and the GAN, for a given configuration, the hyperparameters that had the most impact on the results were the minibatch size and the learning rate. When training differentially private models, the values of the hyperparameters were the same as those of its corresponding non private configuration. To guarantee the privacy, both the autoencoder and the GAN were trained from scratch (i.e. their non private counterpart were not used at any point).

Appendix B.4. Configurations

As stated previously, different configurations were tested to see the impact of changing the types of some features. These configurations as well as the values of their hyperparameters are listed in Table B.3. Note that, except for the types of the features, the "baseline" and "all_cat" configurations share the same hyperparameters. This is because using the values of the first on the second gave good results.

Table B.3: Hyperparameter values for the different configurations.

Hyperparameters		Configuration		
		baseline	all_cat	bin
Features	DriverAge	Numerical	Numerical	Categorical
	CarAge	Numerical	Numerical	Categorical
	Density	Numerical	Categorical	Categorical
	Exposure	Numerical	Categorical	Categorical
AE	l2 norm clip	0.022	0.022	0.022
	Minibatch size	64	64	128
	Compression dim	25	25	50
	Learning rate	0.01	0.01	0.01
	β_1 (ADAM)	0.9	0.9	0.9
	β_2 (ADAM)	0.999	0.999	0.999
	L2 penalty	0	0	0
GAN	l2 norm clip	0.027	0.027	0.027
	Clip value	0.01	0.01	0.01
	Minibatch size	128	128	128
	Latent dim	25	25	30
	Learning rate	4.5×10^{-5}	4.5×10^{-5}	3.9×10^{-5}
	Discriminator updates	10	10	5
	Alpha (RMSProp)	0.99	0.99	0.99
	l2 penalty	0	0	0

During the training of the autoencoder, the learning rate was reduced by a factor 0.2 when the validation loss reached a plateau (tolerance of 1×10^{-4}) for 1000 iterations (i.e. the patience). Its minimum value was limited to 1/100 of the initial learning rate shown in Table B.3. Two other hyperparameters choices were tested, but are not listed in Table B.3 as they did not improve the results. The first was using a Kaiming uniform initialization of the weights for the AE and GAN instead of the default pytorch initialization. The second was using the ADAM optimizer for the GAN instead of RMSProp. Both optimizers gave almost the same results when using the same random seed.