Improving Error Detection with GAN Models

Joshua Lemmon
University of Waterloo
jmlemmon@uwaterloo.ca

Abstract

A common issue in classification tasks is lack of training data and imbalanced data sets. In the past it has been shown that data augmentation methods can help circumvent this issue. In this project, I propose the use of GAN models to help improve the task of error detection in databases. Some empirical experimental results show that GAN generated data does improve error detection performance in general using classifiers not completely suited to the task.

1 Problem Statement

Error detection has been a prominent research topic in the field of Data Cleaning for years. (Abedjan et al., 2016) However, a huge issue with training models to detect potential errors in a data set stems from the lack of training data. Typically, the ratio of clean to dirty tuples in a data set is very low. This results in issues when training models for detection due to an imbalance of classes. Recently, sophisticated machine learning models have been designed that learn representations of errors and performs data augmentation to generate more examples of erroneous data. (Heidari et al., 2019) In machine learning literature, a common architecture used to generate training data is the Generative Adversarial Network (GAN). (Goodfellow et al., 2014) This project explores the possibility of using a GAN, called errorGAN, to enhance the performance of error detection models through data generation.

2 Generative Adversarial Networks

A GAN is a neural network model consisting of two sub-models: a generator and a discriminator. Both of these models are typically a feed-forward multiplayer perceptron.

2.1 Generator

The generator model G accepts some random noise, z, as input and outputs a vector in the size of the data space.

2.2 Discriminator

The discriminator model D accepts input, x, in the size of the data space and outputs a scalar value corresponding to the certainty that x is a generated sample.

2.3 Training GANs

When training a GAN, both G and D are trained simultaneously. The training objective of D is to maximize the probability accuracy of predicting whether an input is a generated sample or a real sample. G is trained to minimize $log(1 - D(G(z))$, e.g. minimizes the ability of the discriminator to differentiate between a real and generated sample.

2.4 errorGAN for Error Generation

Traditionally GANs were designed for use with image data, using convolutional layers in the generator and discriminator. errorGAN replaces the convolutional layers with dense layers. The overall model architecture of errorGAN can be seen in (fig. 1). The generator model (fig. 2) accepts a noise vector of size 1024 as input. It has two dense layers of size 128, each with a ReLU activation function and batch normalization. The output of the first layer is concatenated to the output of the second before being input into the final layer that will output a sample tuple. The discriminator model (fig. 3) also consists of two dense layers, one of size 256 and one of size 128, with ReLU activation functions. The final layer outputs the discriminator's prediction. All models use the Adam optimizer.

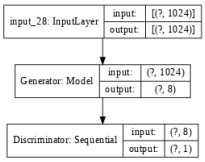


Figure 1: errorGAN model overview

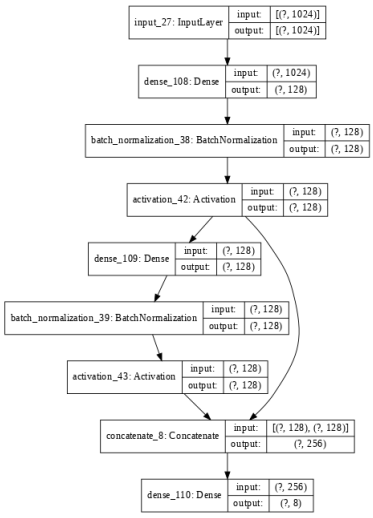


Figure 2: errorGAN generator architecture

3 Data Sets

Three data sets were used for this project: *Person*, *Titanic* and *Credit Card*.

3.1 Person

The *Person* data set is a synthetic data set generated using the ToXgene XML data generator (Barbosa et al., 2002) converted to tuple form. The data set consists of 8 attributes: f_name, l_name, email, age, salary, height, interest, country.

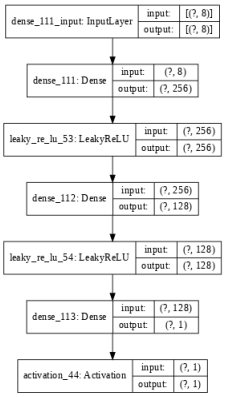


Figure 3: errorGAN discriminator architecture

Data Set	# Cols	# Rows	Err. %
Person (P)	8	10000	5
Titanic (T)	12	1309	10
Credit Card (CC)	31	284807	5

Table 1: A summary of the data sets used

3.2 Titanic

The *Titanic* data set (Kaggle, 2015) is a publicly available data set found on Kaggle that details information about passengers on the Titanic. It consists of 12 attributes, some numeric and some categorical. The original data set is split into two parts, train and test, and they are merged into one file during pre-processing.

3.3 Credit Card

The *Credit Card* data set (ULB, 2016) is another Kaggle data set that consists of anonymized credit card info for use in fraud detection. It consists of 31 columns, all numeric.

3.4 Pre-Processing

For each data set, it is assumed that the data set is clean. A script is run on each clean data set, with takes a portion of the data set and adds noise. The script then adds a column *is_dirty* to denote whether the row has been dirtied or not. The attributes for a row are randomly dirtied. The attributes are dirtied in the following ways.
Numeric Value:

- Value is negated
- Value is multiplied by a factor of 10
- Value is divided by a factor of 10

Categorical Value:

- Value has a random character inserted
- Value has a random character deleted
- Randomly swap characters in the value
- Randomly replace a character in the value

The errors induced in the data sets attempt to replicate possible systematic errors. A summary of the data sets used and the amount of induced errors can be seen in table 1.

3.5 Scaling and Encoding

I also performed some scaling and encoding on the data after loading it into the notebook.

Min-Max Scaling For numerical attributes, I performed min-max scaling. I found that scaling the numeric attributes improved the output of the generator, as without scaling the output attributes of the generator were far too large. Since the CC data set was already encoded numerically I experiment with both a scaled and unscaled version of the data set.

Categorical Encoding I treated all string attributes as categorical values, and encoded them by mapping each unique value to an integer in a column-wise fashion. I experimented with performing one-hot encoding as well, but found that one-hot encoding didn't improve results and only increased run time due to the larger vector size.

4 Experiments

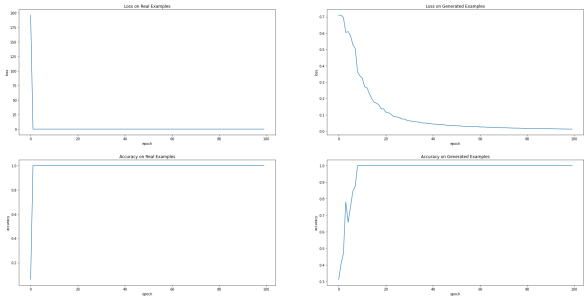
To determine the effect of GAN generated data on the performance of an error classifier I perform a series of tests. I first split the data set into a train and test set. I then train errorGAN on the training set. Once the GAN is trained I then create a series of SVMs. I then train the SVM on the training set with varying amounts of generated erroneous data, and then evaluating the SVM on the test set. When training the GAN I run two separate training batches on the discriminator, one with the real data and one with the generated data, so that I could see the training metrics on both types of data. The code and some of the datasets (CC was too large) can be found on my github.[1]

4.1 Experimental Setup

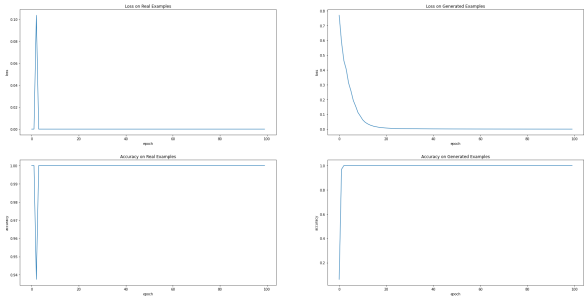
All experiments were performed in an iPython Notebook run on Google Colab. When training the GAN I set the number of epochs to 100 and the batch size to 32. I set the size of the random noise vector to be 1024. When partitioning the data set I did a 70/30 train/test split. The error classifier I used was the default C-SVM from the *sklearn* package.

4.2 Results

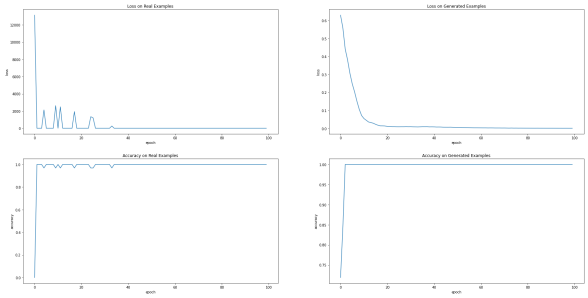
Plots of training loss and accuracy on each data set can be seen in figure 4.



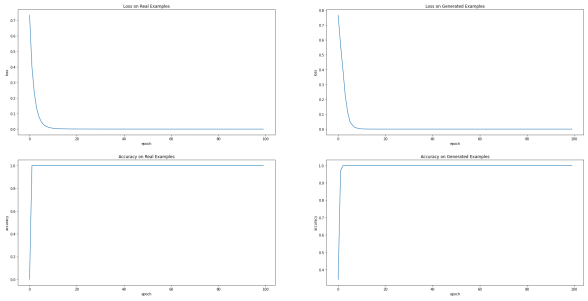
(a) *Person (P)*



(b) *Titanic (T)*



(c) *Unscaled Credit Card (CC)*



(d) *Scaled Credit Card (CC)*

Figure 4: GAN Training Metrics.

4.3 Training Loss

The training loss curve on generated samples stays fairly consistent across all data sets; i.e. it follows the expected loss curve. There are some strange result with the training loss on real samples; the P data set has an expected loss curve, but the T and CC data sets have some unusual spikes before bottoming out at a loss of 0. I found that this happens when I do not use min-max scaling on the CC data set. When I do scale CC, the loss curve looks as expected for real data, however it creates differ-

[1] https://github.com/joshualemmon/db-error-detection-gan

ent precision and recall values when training the SVMs later. Interestingly enough, it seems like the accuracy curve mirrors the loss spikes. I believe the spikes are due to the use of the Adam optimizer, I experimented a bit with using RMSProp as well but found Adam to perform the best overall.

4.4 Training Accuracy

In all cases the discriminator accuracy on both real and generated converges to 100% very quickly. For GAN architectures this is not a very good result, ideally the discriminator should have around a 50% accuracy on both types of samples. The fact that the discriminator has perfect classification means that it is greatly overpowering the generator. In an effort to prevent this, I implemented Batch Normalization and skip connections in the generator, based off of the same technique being used in the MedGAN paper. (Choi et al., 2017) Unfortunately this only marginally helped, as the discriminator still overpowers the generator in training. However despite this, the generated data generally has a positive effect on the performance of the SVM classifiers even after being overpowered during training.

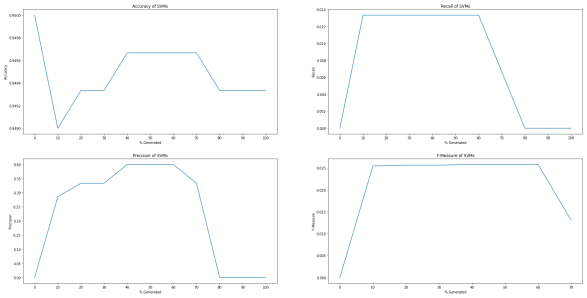


Figure 5: Performance of SVM on Person data set

4.5 Person Results

Figure 5 shows the results of various SVMs trained on increasing amounts of generated data. As you can see, the base model without generated data achieves an accuracy of 95%, with recall, precision and F-measure values of approximately 0. After generating and adding only 10% (700) more erroneous samples, the recall and precision immediately improve. The recall only has a marginal improvement of 0.013, however the precision improves by 0.28. As more samples are added, the recall levels out but the precision increases to a peak of 0.39. After increasing the size of the data by 80% the SVM started running into some issues training, resulting in some NaN values, which is why the F-measure graph stops at 70%. This is because once 80% of generated data was added the SVM started predicting everything as non-erroneous, so the recall and precision became 0.

There is a minor accuracy hit of 0.5% at peak recall and precision (60%), however since the data set has an inherent class imbalance the decrease in accuracy is an acceptable loss. It is much more important that the recall and precision increases. Unfortunately the recall improvement is very marginal, since of the metrics used for error detection recall is the most useful; i.e. we want to find as many errors as possible, and it is okay to get false positives since those can be sorted out using domain knowledge later. The precision increases by a large amount however, which is a positive sign for the GAN method.

While the results are not very promising in terms of improving recall, I believe that this is due to the amount of categorical attributes in the data set. It stands to reason that since SVMs are a more simplistic classification model, it would struggle with determining erroneous string values as opposed to numeric errors.

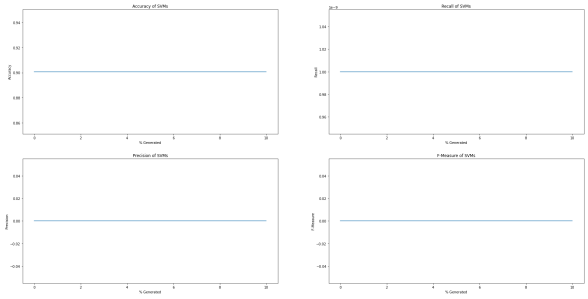


Figure 6: Performance of SVM on Titanic data set

4.6 Titanic Results

Figure 6 shows the results of the SVM on the Titanic data set. Unfortunately, both the base SVM and the SVMs with generated data do not perform well at all on this data set. All metrics are 0 no matter the amount of generated data added. Attribute-wise it is similar to Person, as both have a mix of numeric and categorical values. Therefore, I believe this is simply due to the fact that the Titanic data set is so small, as it only has 1309

tuples, versus the 10000 in the Person data set. I tried to offset this by increasing the percentage of errors induced to 10% instead of 5%, but it did not make a difference.
From this it seems that generated data is not sufficient for all imbalanced data sets, as the initial data set must be large enough to properly train the GAN on in the first place. Perhaps this issue can be circumvented by using a separate GAN to learn the clean data set and generate more samples of clean data that can then be dirtied. However it might be less than ideal to learn error features from previously generated data with its own implicit noise.

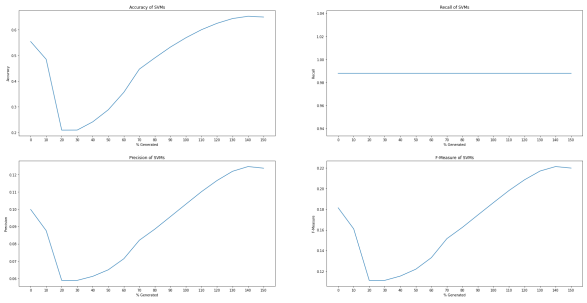


Figure 7: Performance of SVM on Credit Card data set (unscaled)

4.7 Credit Card Results

Of the three data sets, CC shows the most promising results. Figure 7 shows the results of the unscaled CC data set. I extended the amount of generated data to 150%, since in my initial tests it looked like the performance increase would continue to increase, which proved to be true. The performance of the GAN data is quite promising on the unscaled CC, as after a brief accuracy and precision drop off it can be seen that the accuracy increases from the base of 56% to a max of 66% and the precision has a minor improvement from 0.100 to 0.125. The recall stays a consistent 99% throughout all amounts of added data, which shows for the unscaled, numeric CC data the SVM is good at detecting the errors, and the additional error data allows it to guess non-erroneous tuples more consistently. I also tested the effects of generated data on the scaled CC data set, which can be seen in figure 8. The scaled CC results differ in the fact that with only a small amount of added data (20%), the accuracy and precision of the SVM increases by a significant amount, with only a small decrease in recall. The accuracy increases from

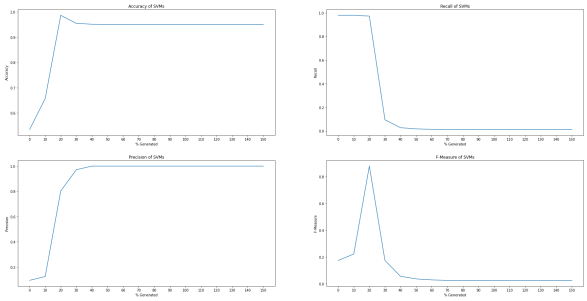


Figure 8: Performance of SVM on Credit Card data set (scaled)

56% to 99%, the precision increases from 0.1 to .82%, while the recall only suffers a minor drop of 1%. Past 20% added data the precision continues to increase, however the recall drops by a large margin and the accuracy drops by a smaller amount. From this it can be seen that even if the data being worked on is already anonymized and encoded, it still makes sense to perform min-max scaling before training the GAN. Overall, it seems like if a small recall loss is acceptable, then the performance increase in precision and accuracy seems worth it.

5 Related Work

While using GANs for data generation is not a new concept, they are generally not used for the specific task of creating intentionally erroneous data.

5.1 RenderGAN

RenderGAN (Sixt et al., 2016) is a GAN network built for generating labelled image data for use in object detection. This is similar in concept to my project, however they are using GANs on the more traditional data format of images, and they are not trying to create erroneous data to train on. Their results seem to coincide with mine a bit, as they show that models trained on generated data have increased performance relative to models trained solely on the base data set.

5.2 medGAN

As mentioned before, medGAN is a GAN model built to generate realistic medical database tuples. In this case their data format is similar to mine, however like RenderGAN they are aiming to create non-erroneous data. They again show that tasks using generated records are comparable to or better than the original data.

6 Future Work

I have a number of ideas as to how this project can be expanded on in the future.

6.1 Improving Base Classifier

My project uses an SVM as the classifier, which I knew from the start would not be the best performing classifier. However since my goal was to test whether performance could be improved in general, this was sufficient. The next step would be to see if this GAN method could improve the performance of more sophisticated error detection models like HoloDetect.

6.2 Implementing More GAN Optimizations

GAN models are generally regarded as difficult models to train, due to the balance that should be maintained between the generator and discriminator. While I implemented some optimizations in my model, the generator was still quickly overpowered by the discriminator. In spite of this, the weakened generator data still induced a positive effect on the error detection. This leads me to believe that if more research on the errorGAN model was done, further optimizations would improve the balance between the generator and discriminator, and thus result in even more effective data to train with.

6.3 Different Encoding Methods

From my results it is clear that the errorGAN model struggles improving data sets with many categorical values. The two basic methods I used, categorical encoding and one-hot encoding, did not differ much in results. However, perhaps more sophisticated encoding methods such as word2vec (Mikolov et al., 2013) or BERT (Devlin et al., 2018) could more sufficiently encode the categorical attributes and improve the error detection. medGAN uses some encoder/decoder layers in their model, so improving the encoding method seems to be a good next step.

7 Conclusion

Data augmentation through the use of GAN models seems to be a promising method for the task of error detection. Even the performance of simple linear classifiers such as SVMs can be enhanced using data generated from a GAN model, especially on data sets where there are many numeric attributes. While the error detection improvements were lackluster in some instances, the important takeaway is that the generated data does improve performance in general.

References

Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (Jan 2016), 993–1004. https://doi.org/10.14778/2994509.2994518

Denilson Barbosa, Alberto Mendelzon, John Keenleyside, and Kelly Lyons. 2002. ToXgene: An extensible template-based data generator for XML. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data - SIGMOD 02* (2002). https://doi.org/10.1145/564691.564769

Edward Choi, Siddharth Biswal, Bradley A. Malin, Jon Duke, Walter F. Stewart, and Jimeng Sun. 2017. Generating Multi-label Discrete Electronic Health Records using Generative Adversarial Networks. *CoRR* abs/1703.06490 (2017). arXiv:1703.06490 http://arxiv.org/abs/1703.06490

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:stat.ML/1406.2661

Alireza Heidari, Joshua McGrath, Ihab F. Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 829–846. https://doi.org/10.1145/3299869.3319888

Kaggle. 2015. *Titanic: Machine Learning from Disaster*. https://www.kaggle.com/c/titanic

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:cs.CL/1301.3781

Leon Sixt, Benjamin Wild, and Tim Landgraf. 2016. RenderGAN: Generating Realistic Labeled Data. *CoRR* abs/1611.01331 (2016). arXiv:1611.01331 http://arxiv.org/abs/1611.01331

Machine Learning Group ULB. 2016. *Credit Card Fraud Detection*. https://www.kaggle.com/mlg-ulb/creditcardfraud