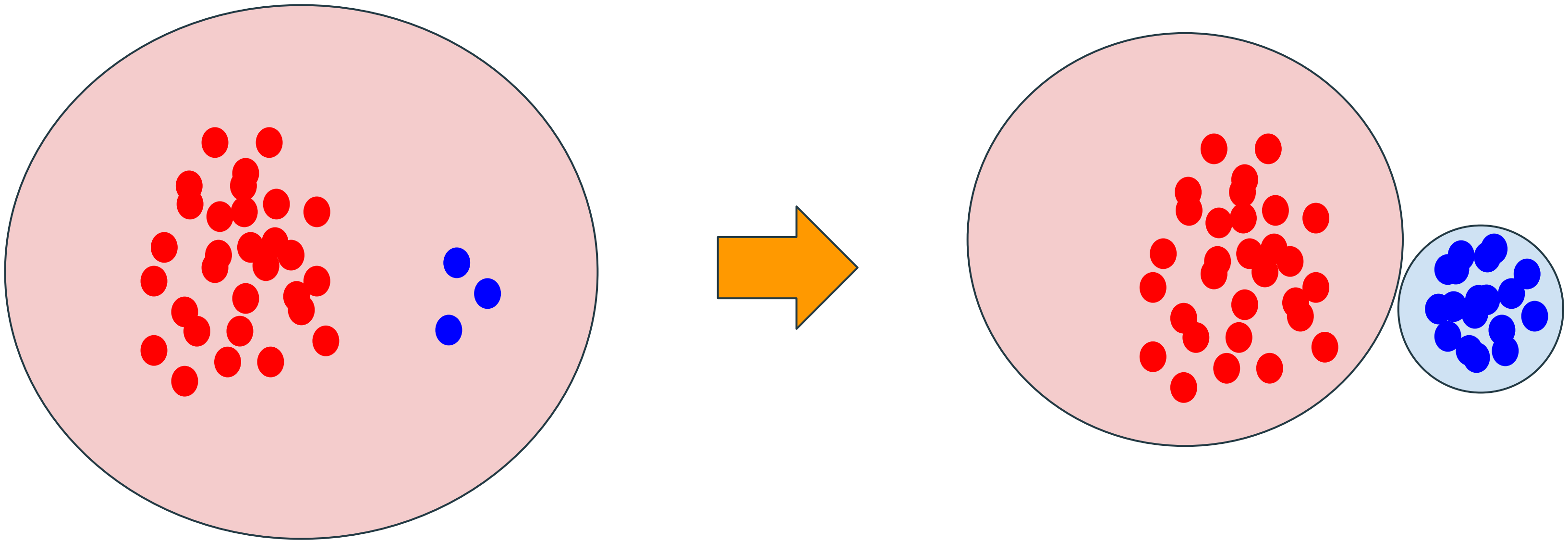
INSIGHT

RivalGan
Synthesizing business data to balance classes

Yves Greatti

Machine Learning with imbalanced data sets is difficult



RG

Imbalanced classes are impediments in many fields

Credit card fraud detection

- Credit card fraud cost US consumers **$16 billion** in 2016
- **42-47%** of flagged transactions are manually reviewed

Medical research

- For privacy concerns and cost of clinical trials no easy access to health care records

RG

Credit card fraud detection dataset is completely imbalanced

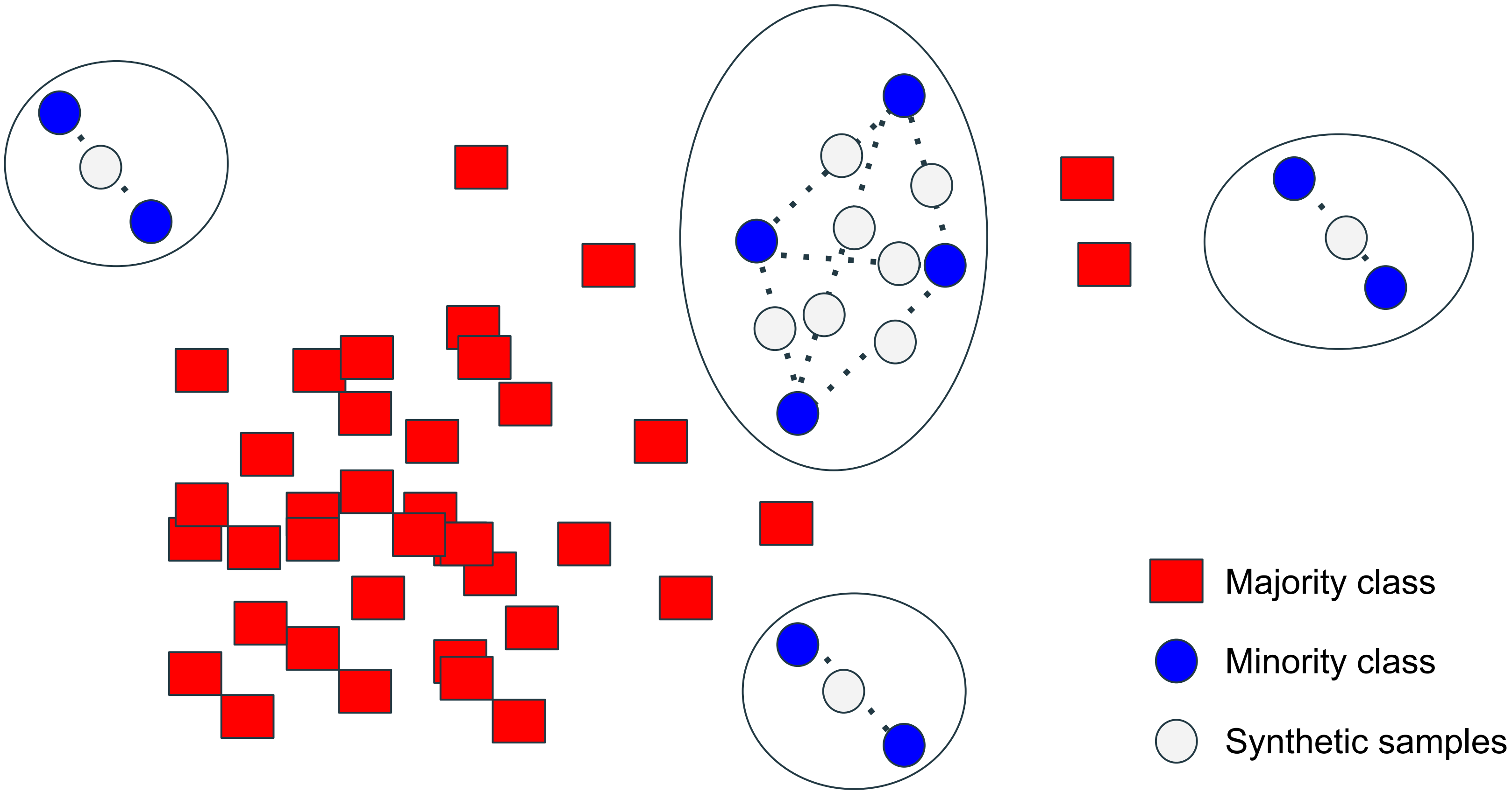
284,807 transactions

99.83%
Non frauds

0.17%
Frauds

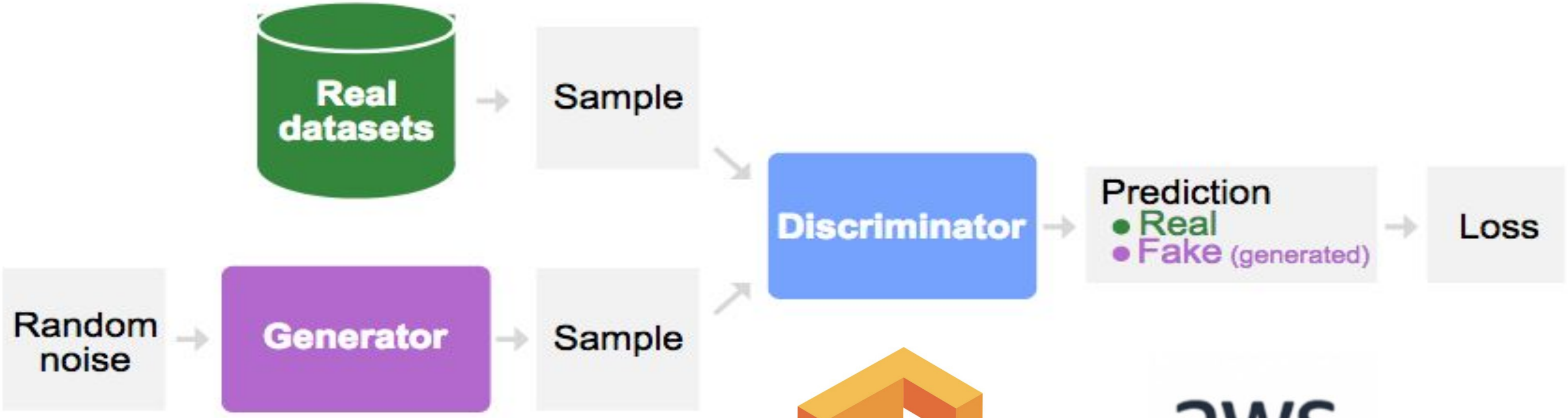
RG

SMOTE: Synthetic Minority Over Sampling Technique



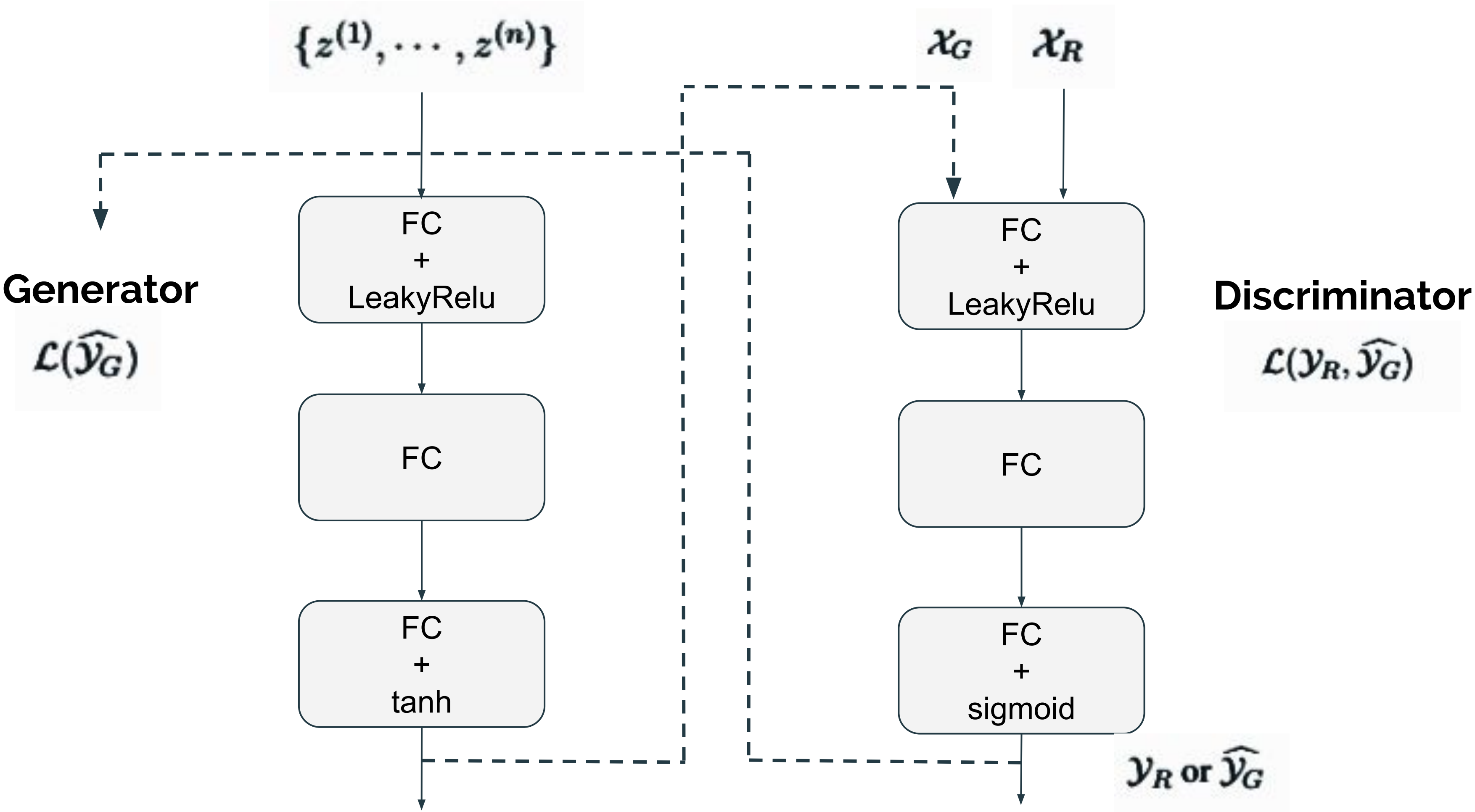
RG

GAN is a two-player game, between the discriminator and the generator

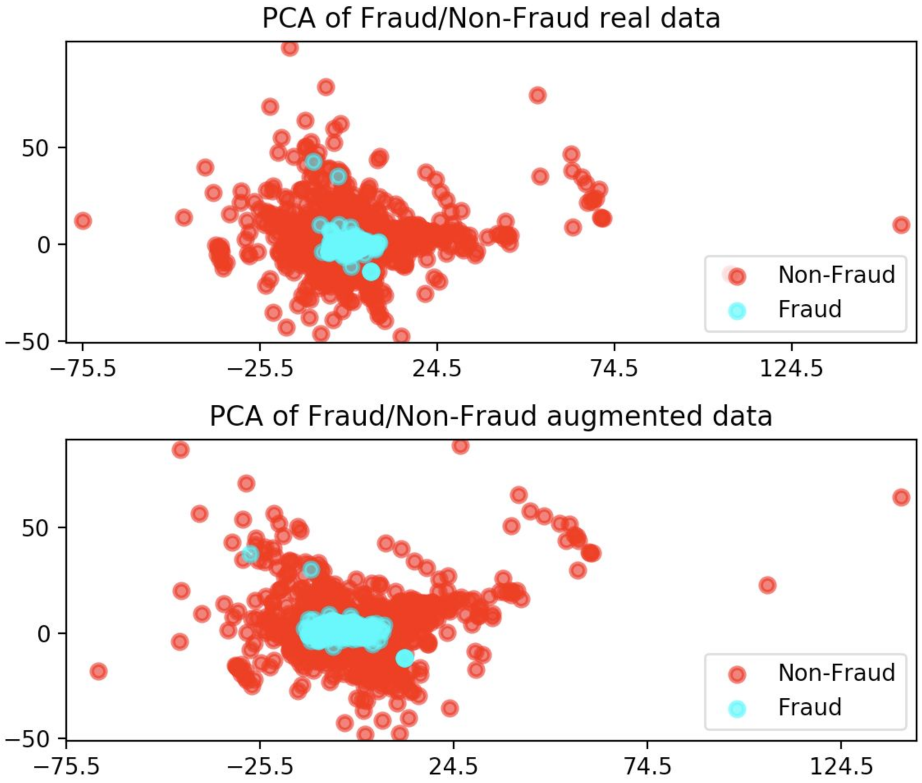


RG

GAN is a two-player game architecture



Data distributions are similar



Classifier using traditional synthetic sampling technique (SMOTE) does not perform better

Linear SVM	F1 score: 0.064

Traditional approach (SMOTE)	F1 score: 0.078 (< 8%)

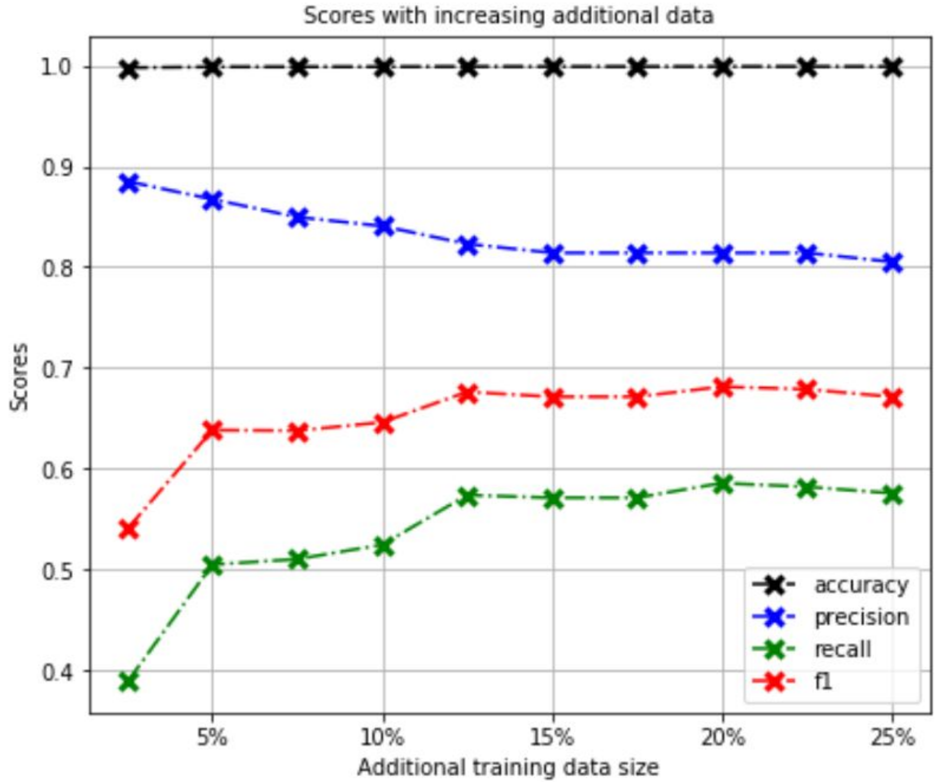
RG

Classifier using generated data has astounding better performance (relative improvement of 4 times)!

Linear SVM	F1 score: 0.064
Traditional approach (SMOTE)	F1 score: 0.078 (< 8%)
My approach (GAN)	F1 score: 0.44 (**44%**)

RG

F1 score keeps improving as we train with more synthetic data



False positives are reduced by about 90%

Imbalanced dataset

True label \ Predicted label	Non Fraud	Fraud
Non Fraud	68098	2991
Fraud	10	103

Balanced dataset

True label \ Predicted label	Non Fraud	Fraud
Non Fraud	70770	319
Fraud	13	100

Cost analysis

	Traditional approach	My approach
False positive rate	4.2%	0.4%
False positive	2991	319
Estimated cost of manual review	$42-$70	$42-$70
	$51k - $90k	$7k - $12k

$80,000 in savings!

RG

Available on or

Search projects

rivalgan 0.4

```
pip install rivalgan
```

RIVALGAN

RG

Usage

```
$ python pipeline -h

usage: pipeline.py [-h]
                   [--CLASSIFIER {Logit,LinearSVC,RandomForest,SGDClassifier,SVC}]
                   [--SAMPLER {SMOTE,SMOTETomek}]
                   [--AUGMENTED_DATA_SIZE AUGMENTED_DATA_SIZE]
                   [--TOTAL_TRAINING_STEPS TOTAL_TRAINING_STEPS]
                   [--GEN_FILENAME GEN_FILENAME]
                   [--train_classifier TRAIN_CLASSIFIER]
                   [--classifier_scores CLASSIFIER_SCORES]
                   [--generate_data GENERATE_DATA]
                   [--compute_learning_curves COMPUTE_LEARNING_CURVES]
                   [--aug_model_scores AUG_MODEL_SCORES]
                   [--plot_augmented_learning_curves PLOT_AUGMENTED_LEARNING_CURVES]
                   [--generate_distribution_plots GENERATE_DISTRIBUTION_PLOTS]
                   [--compare_scores COMPARE_SCORES]
                   [--random_dataset RANDOM_DATASET]
                   [--retrieve_real_data_generated_data RETRIEVE_REAL_DATA_GENERATED_DATA]
```

Usage

```
$ python pipeline -h

usage: pipeline.py [-h]
                   [--CLASSIFIER {Logit,LinearSVC,RandomForest,SGDClassifier,SVC}]
                   [--SAMPLER {SMOTE,SMOTETomek}]
                   [--AUGMENTED_DATA_SIZE AUGMENTED_DATA_SIZE]
                   [--TOTAL_TRAINING_STEPS TOTAL_TRAINING_STEPS]
                   [--GEN_FILENAME GEN_FILENAME]
                   [--train_classifier TRAIN_CLASSIFIER]
                   [--classifier_scores CLASSIFIER_SCORES]
                   [--generate_data GENERATE_DATA]
                   [--compute_learning_curves COMPUTE_LEARNING_CURVES]
                   [--aug_model_scores AUG_MODEL_SCORES]
                   [--plot_augmented_learning_curves PLOT_AUGMENTED_LEARNING_CURVES]
                   [--generate_distribution_plots GENERATE_DISTRIBUTION_PLOTS]
                   [--compare_scores COMPARE_SCORES]
                   [--random_dataset RANDOM_DATASET]
                   [--retrieve_real_data_generated_data RETRIEVE_REAL_DATA_GENERATED_DATA]
```

Next steps

- Add an autoencoder before the GAN to work in the latent space (adversarial autoencoder)

- Use different scoring functions (Micro/Macro F1, Matthews Coefficient)

- Provide an UI front-end in addition of the existing API

Yves Greatti

NYU | COURANT INSTITUTE OF MATHEMATICAL SCIENCES

LABORATOIRE D'INFORMATIQUE FONDAMENTALE de Marseille

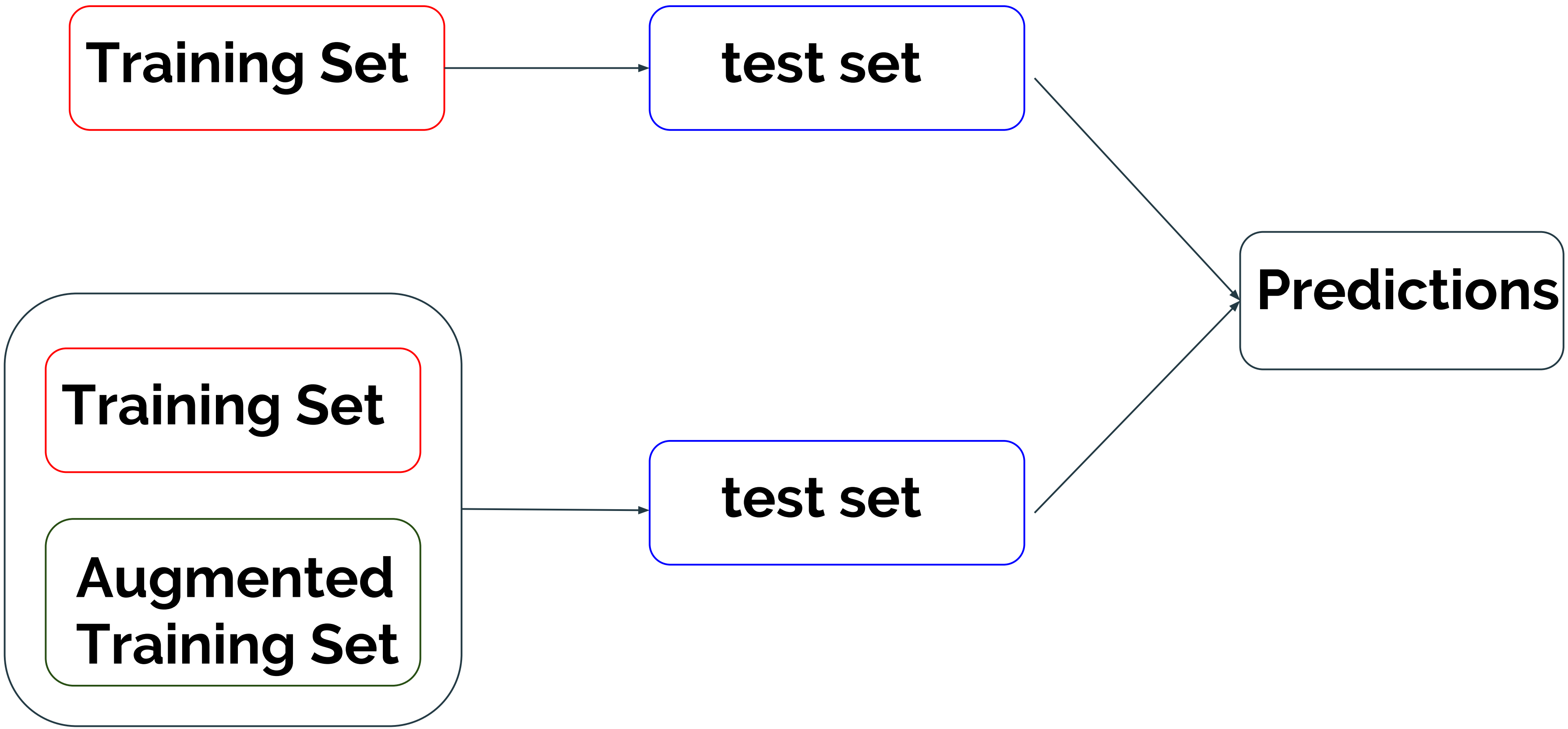
Laboratoire d'Informatique Fondamentale

UNIVERSITÉ DE LYON

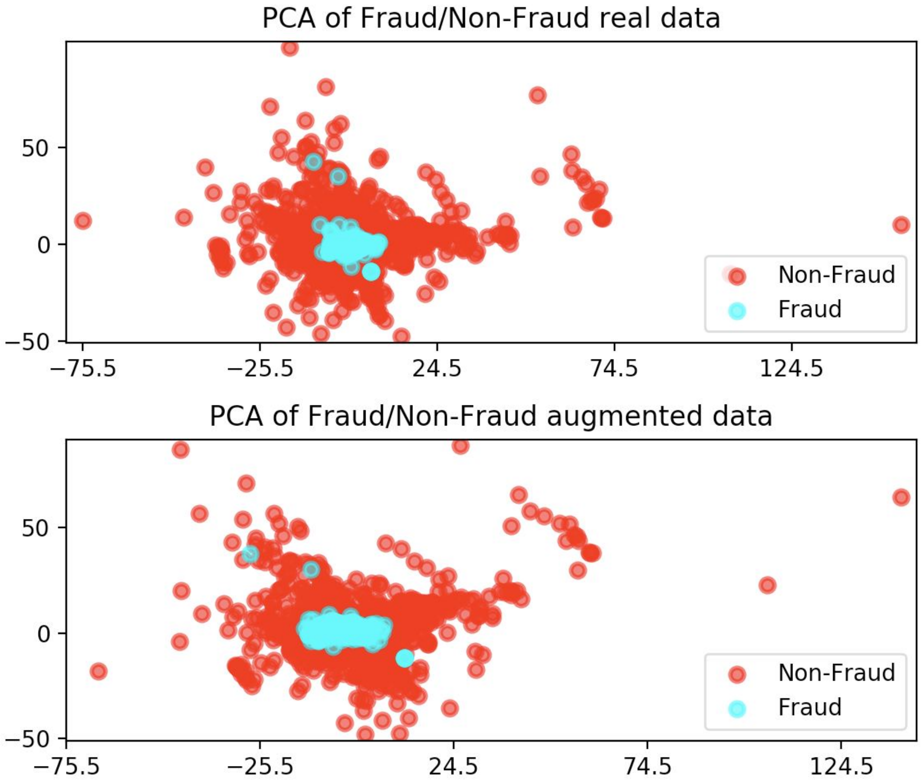
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

ICE/NYSE Team Lead

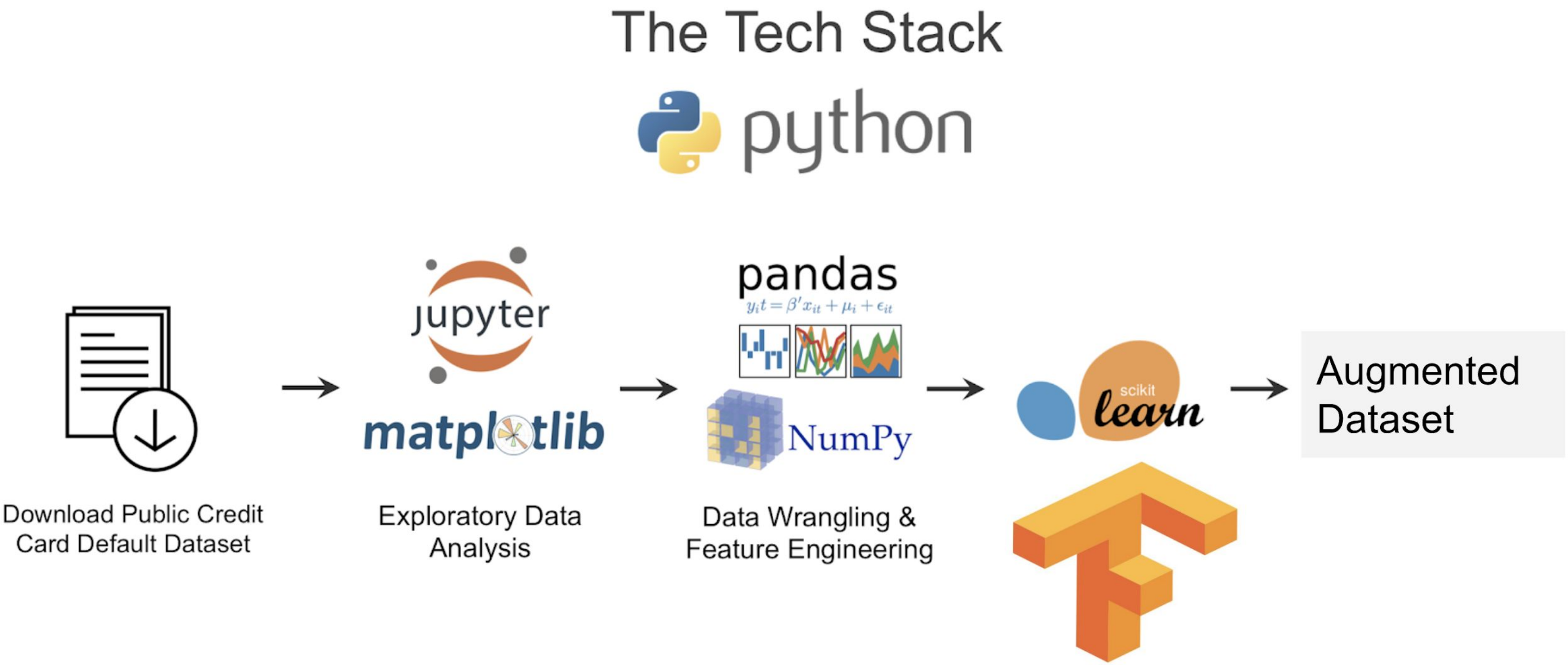
EXTRA SLIDES


Training Set
test set
Predictions
Training Set
Augmented Training Set
test set

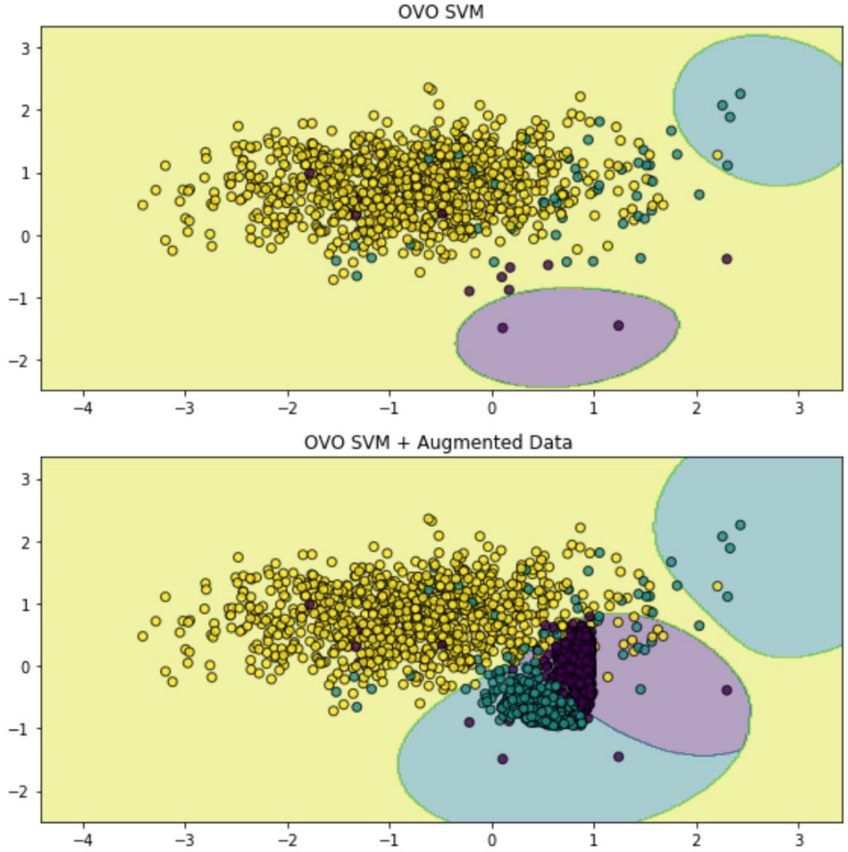
Data distributions are similar



Current pipeline is a mixed of different libraries and running on AWS



On a random data set, balancing the data set improves decision boundaries



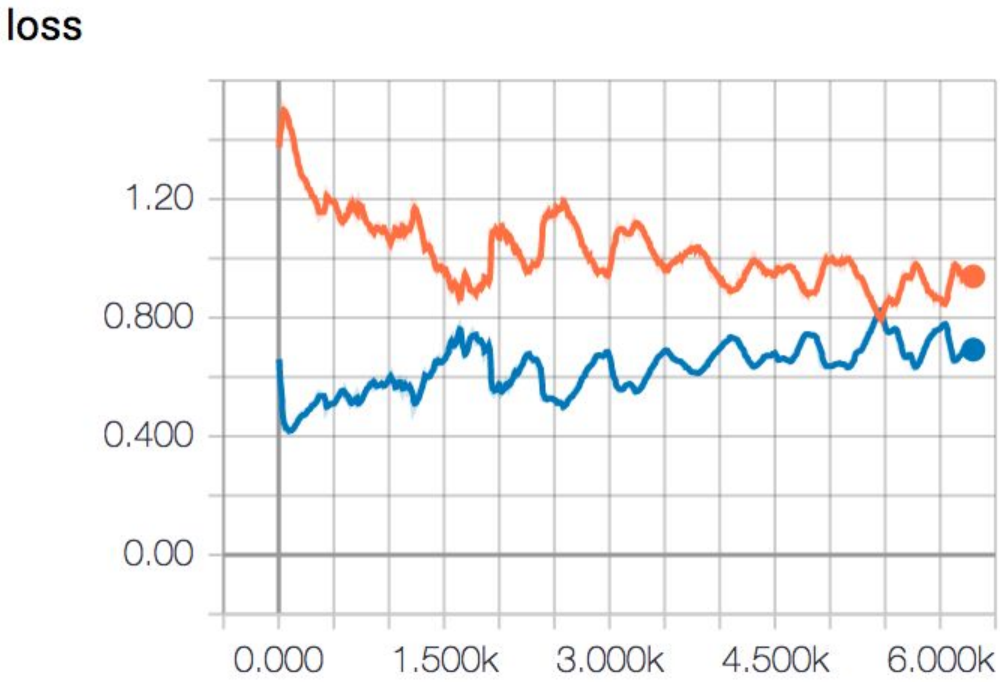
30 Features

V1 through V28

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
0	-0.304032	-0.942959	0.981965	0.960006	-0.846038	-0.965205	0.990126	-0.991626	-0.938985	0.956888
1	-0.275078	-0.978785	0.994081	0.959790	-0.602292	-0.903874	0.812272	-0.992481	-0.936116	0.957471
2	0.021408	-0.970741	0.966797	0.970045	-0.668063	-0.878281	0.939800	-0.939541	-0.936466	0.885240

- *Time*
- *Amount* = transaction amount
- *Class* = variable to predict 1 in case of fraud or 0 otherwise

Equilibrium is reached



Discriminator loss

Generator loss

Quality of the generated data for the minority class does not seem to be related to the GANs architecture

Vanilla GAN	Accuracy score: 0.99 F1 score: 0.44
Wasserstein GAN	Accuracy score: 0.99 F1 score: 0.41
Improved Wasserstein GAN	Accuracy score: 0.99 F1 score: 0.41
Least Square GAN	Accuracy score: 0.99 F1 score: 0.39

The problem is clearly identified

Binary classification with strong class imbalances

- Poor model performance: high accuracy but F1 close to zero

- Model prediction is not useful

Existing ML solutions are not satisfying

- Adaptive ML algorithms like AdaBoost sensitive to noisy data

- SMOTE,ADASYN, RUS could adversely affect the features distribution patterns

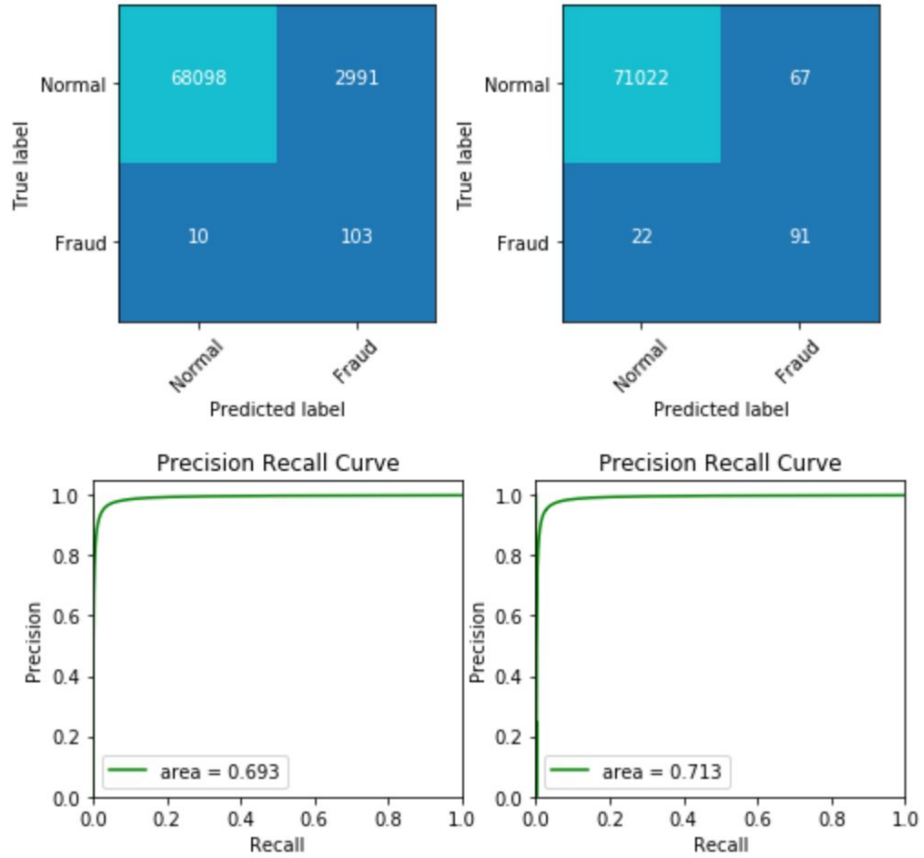
Using GANs makes sense because...

1. GANs are capable of learning the prior distribution of the real data.
2. Augmented data is a proportional scaling of the features' distributions
3. Overfitting is reduced.

Output

```
------------- Reading data --------------

Loading data from /home/ubuntu/insight/data/creditcard.engineered.pkl
Shape of the data=(284807, 31)
Head:
        Time        V1        V2        V3        V4        V5        V6  \
0 -2.495776 -0.760474 -0.059825  1.778510  0.998741 -0.282036  0.366454
1 -2.495776  0.645665  0.177226  0.108889  0.326641  0.047566 -0.064642
2 -2.495729 -0.759673 -0.946238  1.240864  0.277228 -0.418463  1.425391

         V7        V8        V9  ...         V21       V22       V23  \
0  0.234118  0.091669  0.343867  ...   -0.027953  0.392914 -0.259567
1 -0.078505  0.077453 -0.237661  ...   -0.405091 -0.908272  0.228784
2  0.775964  0.247431 -1.420257  ...    0.456138  1.094031  2.092428

        V24       V25       V26       V27       V28    Amount  Class
0  0.111992  0.253257 -0.396610  0.399584 -0.090140  1.130025      0
1 -0.569582  0.329670  0.267951 -0.031113  0.069997 -1.138642      0
2 -1.155079 -0.649083 -0.291089 -0.171222 -0.263354  1.695499      0

[3 rows x 31 columns]
Number of frauds in training data: 379 out of 213605 cases (0.1774303036% fraud)
Number of frauds in test data: 113 out of 71202 cases (0.1587034072% fraud)
Number of features=30

------------- Training classifier --------------


Training 30 features with classifier SGDClassifier
Time elapsed to train:  0:00:00.34
Saving SGDClassifier in /home/ubuntu/insight/cache/SGDClassifier_Fraud.pkl
No sampler to train
```

Linear SVM Precision/Recall curves are improved with augmented data set



T-sne of the data distributions