Synthetic Data: Quality

We have encountered several methods for generating Synthetic Data.

A natural question to ask

- How do we know that Synthetic examples are "good enough" ?

Our Generative models were trained so as to minimize a Loss function

- Generate a synthetic "equivalent" for each training example
- We used distance measures to quantify "equivalent"
    - KL
    - JSD
    - EMD (Wasserstein)

Is creating a synthetic replica for each training example really sufficient (i.e., "good enough") ?

We discuss metrics that may be used to evaluate the quality of synthetic data.

What is ”good enough” ?

Let us hypothesize that there is some true (but unknown) distribution p_{data} of *real* examples.

It should be the goal of our Generative Model to learn to create a distribution p_{model} of *synthetic* examples such that

$$p_{\text{model}} \approx p_{\text{data}}$$

That is **not** what the Loss function is doing

- Two distributions can be identical
- Without having samples drawn from each being identical

For example: two fair coins, each flipped a large number of times, will not likely produce the same sequence of results.

Moreover:

- Distributions that are "close" on an example by example basis
- May be sufficiently different as to not be "good enough" for some purposes
    - Mean value of each feature of a near replica synthetic example off by some constant amount relative to the source training example
        - biased estimated of the return of each stock

It would seem that "good enough" is not a universal measure

- May be task specific
    - Dependent on how the synthetic data will be used

It is also likely that, for any given task, there are a number of metrics that are desirable.

Quality: Measures

We will examine a number of metrics of quality below.

The metrics will compare

- samples drawn from p_{model}
- samples drawn from p_{data}

in order to address the question

$$p_{\text{model}} \approx p_{\text{data}}$$

Diversity

A *visual* comparison of the distributional properties of the real and synthetic samples.

We can compare the distributional properties of the two samples to see if they are "close enough".

- the moments of each sample's n features
- the cross-sectional relationship among the n features

If the examples are higher dimension (e.g., timeseries) we can examine the higher dimensional properties.

For example, if an example is a timeseries of daily attributes

- we can examine the time relationships

We would hope that a high quality synthetic sample demonstrated similar relationships as a real sample.

If timeseries $\mathbf{x} = \mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(T)}$ is a sequence of n attributes over T time steps, we can examine

- Autocorrelation Function (ACF)
  - how do lagged values $\mathbf{x}_{(t-k)}$ correlate with $\mathbf{x}_{(t)}$, for various lags k ?
- Partial Autocorrelation Function (PACF)
  - the PACF of lag k is the correlation of $\mathbf{x}_{(t-k)}$ and $\mathbf{x}_{(t)}$ not explained by lesser lags $\mathbf{x}_{(t-k')}$ for $k' < k$
- ARCH
- GARCH
- ARIMA

Fidelity: Classifier based metrics

Can we create a Neural Net Classifier to distinguish between real and synthetic examples ?

- Obtain real and synthetic samples
- Split each sample into a train and test cohort.
- Train a classifier with a combination of (labeled: Real, Synthetic) examples from the real and synthetic cohorts

After training, we evaluate the Classifier out of sample using a combination of (Real, Synthetic) examples from the test split of both samples.

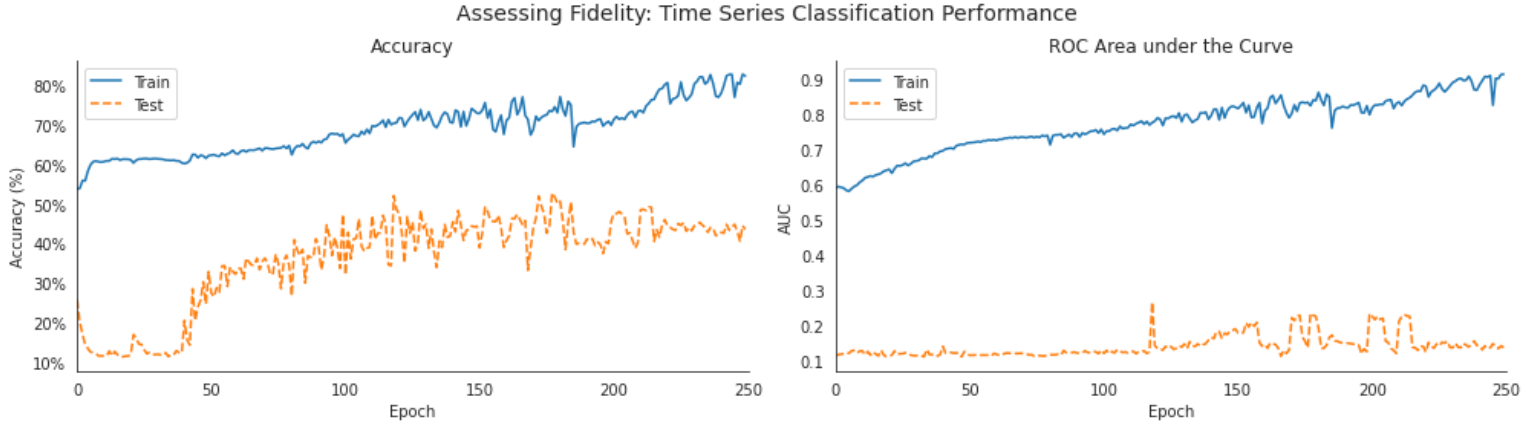
If the out of sample metric is *worse* than the in sample (training)

- then the synthetic sample is "good"
- because a well-trained classifier cannot distinguish real from synthetic examples

To illustrate (left plot)

- a classifier is trained to distinguish between real and synthetic examples (blue `Train` line)
    - high accuracy **in-sample**
- the classifier's **out of sample** accuracy (orange `Test` line) is much worse
- hence: the Classifier **does not distinguish** synthetic from real out of sample

TSTR illustration



Attribution: https://github.com/stefan-jansen/synthetic-data-for-finance/blob/main/02_evaluating_synthetic_data.ipynb

Classifier model-based metrics: Inception

Rather than train a special purpose Classifier for Real/Synthetic classification, we can *re-purpose* a Classifier for a different task.

We illustrate this using a model for Image Classification

- because there are many existing, high-quality models for this task
    - trained on lots of data
    - basis of the ImageNet competition

One can imagine adapting this idea to other types of data for which a high quality classifier is available

- This may be more difficult for tasks on which there are no preexisting, high-quality Classifiers
- Perhaps because of scarcity of training data

Inception is a very successful model for Image Classification.

- Learns to label Images from a large (1000) fixed number of classes

It has been used in several contexts for the purpose of evaluating metrics for synthetic image data.

Inception Score

The idea using a Classifier to compute metrics for synthetic data were proposed in [this paper, Section 4 (https://arxiv.org/pdf/1606.03498.pdf)](https://arxiv.org/pdf/1606.03498.pdf).

Given a synthetic example $\hat{\mathbf{x}}$ the Classifier computes

$$p(\mathbf{y}|\hat{\mathbf{x}})$$

the distribution of labels for the image.

The authors propose that

- synthetic example $\hat{\mathbf{x}}$ is "high quality"
- if $p(\mathbf{y}|\hat{\mathbf{x}})$ is *low entropy*
    - most of the probability is concentrated around a single label
    - Classifier is highly confident

That is: a poor quality image would "confuse" the Classifier, leaving it uncertain as to the correct label.

The authors suggest that high confidence prediction correlates with human judgment.

They also suggest that the set of synthetic $\hat{\mathbf{x}}$ be *diverse*

- not collapse to a single example

We can compute the marginal

$$p(\mathbf{y}) = \int p(\mathbf{y}|\hat{\mathbf{x}} = G(\mathbf{z}))\, \partial \mathbf{z}$$

which represents the distribution of class labels over a set of generated examples.

The authors suggest that the marginal have *high entropy*

- probability spread across many possible labels

That is: different choices of random latent $\mathbf{z}$ give rise to synthetic examples of many different classes.

This is a measure of the *diversity* of the synthetic examples.

They propose a metric, the *Inception Score (IS)* based on the KL divergence for each example

- between $p(\mathbf{y}|\hat{\mathbf{x}}^{(\mathbf{i})})$ and $p(\mathbf{y})$
- want this to be high
    - highly confident prediction $p(\mathbf{y}|\hat{\mathbf{x}}^{(\mathbf{i})})$ for $\hat{\mathbf{x}}$
    - compared to $p(\mathbf{y})$, the "average" (across all examples) prediction

$$
\begin{aligned}
\mathbf{KL}^{(\mathbf{i})} &= \mathbf{KL}(p(\mathbf{y}|\hat{\mathbf{x}}^{(\mathbf{i})})\|p(\mathbf{y})) && \text{for each synthetic } \hat{\mathbf{x}}^{(\mathbf{i})} \\
\bar{\mathbf{KL}} &= \tfrac{1}{m}\textstyle\sum_i \mathbf{KL}^{(\mathbf{i})} && \text{average over synthetic examples} \\
\mathrm{IS} &= \exp(\bar{\mathbf{KL}})
\end{aligned}
$$

It thus combines the high quality and diversity metrics.

Here is a tutorial with code (https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/) on calculating the Inception Score.

Let's review the code

In [2]:

```
# calculate inception score in numpy
from numpy import asarray
from numpy import expand_dims
from numpy import log
from numpy import mean
from numpy import exp

# calculate the inception score for p(y|x)
def calculate_inception_score(p_yx, eps=1E-16):
    # calculate p(y)
    p_y = expand_dims(p_yx.mean(axis=0), 0)
    # kl divergence for each image
    kl_d = p_yx * (log(p_yx + eps) - log(p_y + eps))
    # sum over classes
    sum_kl_d = kl_d.sum(axis=1)
    # average over images
    avg_kl_d = mean(sum_kl_d)
    # undo the logs
    is_score = exp(avg_kl_d)
    return is_score
```

Given `p_yx`

- Python 2D array representing $p(\mathbf{y}|\hat{\mathbf{x}})$
- Each row is a different synthetic example $\hat{\mathbf{x}}^{(\mathbf{i})}$
- Each row is a vector of length $||C||$, where C is the set of Classification labels
    - `p_yx[i]` is the probability distribution $p(\mathbf{y}|\hat{\mathbf{x}}^{(\mathbf{i})})$

The routine `calculate_inception_score(p_yx)` calculates the Inception Score (IS) for the set of synthetic examples.

This is illustrated for low quality examples that classified among 3 labels.

- The quality is "low" because each $p(\mathbf{y}|\hat{\mathbf{x}}^{(\mathbf{i})})$ vector has uniform probabilities

In [3]:
```
# conditional probabilities for low quality images
p_yx = asarray([[0.33, 0.33, 0.33], [0.33, 0.33, 0.33], [0.33, 0.33, 0.33]])
```

Analyzing `calculate_inception_score(p_yx)`

- $p(\mathbf{y})$ (the marginal) is computed by averaging across the examples

```
p_y = expand_dims(p_yx.mean(axis=0), 0)
```

- $p(\mathbf{y}|\hat{\mathbf{x}}), p(\mathbf{y})$ both represented as vectors of length $||C||$

KL divergence computed (for each row/example)

```
kl_d = p_yx * (log(p_yx + eps) - log(p_y + eps))
```

- so `kl_d` dimension is $(m, ||C||)$
- it is the $||C||$ additive terms that are summed to get, $\mathbf{KL^{(i)}}$, the KL divergence for each example

```
# sum over classes
sum_kl_d = kl_d.sum(axis=1)
```

Thus elements `sum_kl_d` are m scalars representing $\mathbf{KL^{(i)}}$ for each example i.

Finally, the Inception Score is

- average (over examples i) of $\mathbf{KL}^{(\mathbf{i})}$
- exponentiated

```
# average over images
avg_kl_d = mean(sum_kl_d)
# undo the logs
is_score = exp(avg_kl_d)
```

Let's see the score for the set of "low quality" $p(\mathbf{y}|\hat{\mathbf{x}})$

In [4]:

```
p_yx = asarray([[0.33, 0.33, 0.33], [0.33, 0.33, 0.33], [0.33, 0.33, 0.33]])
score = calculate_inception_score(p_yx)
print(score)
```

```
1.0
```

And similarly, for high quality (low entropy $p(\mathbf{y}|\hat{\mathbf{x}}))$

In [5]:

```
# conditional probabilities for high quality images
p_yx = asarray([[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]])
score = calculate_inception_score(p_yx)
print(score)
```

```
2.999999999999999
```

Frechet Inception Distance (FID)

[This paper, "Experiments" section (https://arxiv.org/pdf/1706.08500.pdf)]

proposes a different use of the Classifier

- as a **feature extractor**

Deep Learning Classifiers have multiple layers, culminating in a Classification Head.

Each successive layer is producing an alternate representation of the raw input features

- until the layer preceding the Classifier head produces a representation on which the Classification Head can succeed

Rather than evaluating synthetic examples on properties of their *raw* representation

- the authors propose using the representation produced by some layer L of the classifier

The belief is that the representation of deeper layers represent increasingly complex *semantic* concepts

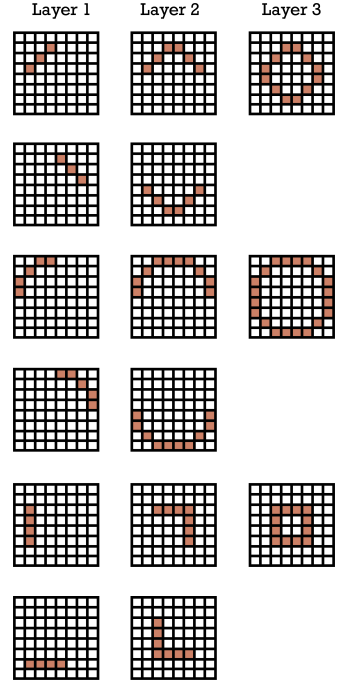
- rather than the *syntactic* concepts of the raw features

For example: consider an image.

Rather than considering an image as a collection of pixels (syntax)

- Consider its representation in deeper layers that recognize "concepts" (semantics): Collections of pixles
    - representing complex shapes

Features by layer



The vector (of length n_L) of layer L's representation can be viewed as a sample from some distribution.

We can form the empirical distribution

- over real examples
- over synthetic examples

The idea is that the two empirical distributions should be similar.

A measure of the dissimilarity of the two distributions is called *Frechet Inception Distance*

- uses the Frechet (Wasserstein, EMD) distance between the two distributions
- under the assumption that the two distributions are Gaussian
    - the Frechet distance is a function of the first two moments

This tutorial (https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/) discusses the computation.

Classifier model-based metrics: discussion

Elevating the representation to semantics from syntax is not without its issues

- The representation of the deepest layers may be too specific to the Classification task
- The representation of shallow layers may not convey enough meaning

- *Adversarial Examples* prove that Classifiers can be fooled with non-meaningful examples
    - calling into question whether using a Classifier as a feature extractor captures aspects of being "real"

Adversarial example of "Speed Limit 45"

So what ? Adversarial Examples 2

Train on Synthetic, Test on Real (TSTR)

This metric is relative to a specific Target Task

- Given a model for the Target task that is trained using a *synthetic* sample
- Compute the test Performance Metric
    - *when the test dataset is real*

Similarly, train a model on a *real* sample and evaluate it on a test *real* sample (TRTR)

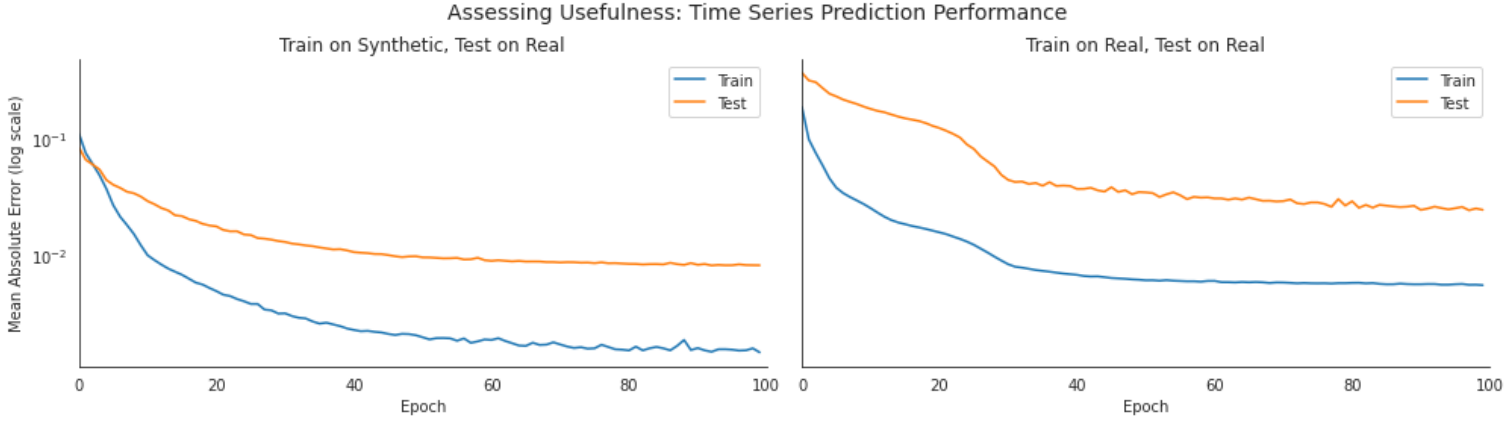
The synthetic sample is "good enough" for the Target task

- if the test Performance Metric (always on a *real* sample)
- is comparable for the model trained on synthetic as the model trained on real

To illustrate: On an out-of-sample dataset of Real examples (Test Real)

- the error of the model trained on Synthetic examples (left)
- is lower than the error of the model trained on Real examples (right)

TSTR illustration



The result shows that synthetic training data may indeed be useful. On the specific predictive task of predicting the next daily stock price for six tickers, a simple model trained on synthetic TimeGAN data delivers equal or better performance than training on real data.

Attribution: https://github.com/stefan-jansen/synthetic-data-for-finance/blob/main/02_evaluating_synthetic_data.ipynb

Other GAN measures

This paper (https://arxiv.org/pdf/1802.03446.pdf) discusses developments of quality metrics for synthetic examples created by GANs.

TimeGAN: Evaluation

Jansen's notebook (https://github.com/stefan-jansen/synthetic-data-for-finance/blob/main/02_evaluating_synthetic_data.ipynb) evaluates the quality of synthetic timeseries created by a TimeGAN, using metrics suggested by the paper's authors.

It is worthwhile looking at the results to get a better feel for the evaluation methods.

In [6]:
```
print("Done")
```

```
Done
```