03_evaluating_synthetic_data

September 29, 2021

Visualize Real and Synthetic Data

1 Imports and Settings

[1]:
```
from pathlib import Path

import numpy as np
import pandas as pd

from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GRU, Dense
from tensorflow.keras.losses import BinaryCrossentropy, MeanSquaredError,␣
 ↪MeanAbsoluteError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import AUC

import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
if gpu_devices:
    print('Using GPU')
    tf.config.experimental.set_memory_growth(gpu_devices[0], True)
else:
    print('Using CPU')
```

```
Using CPU
```

[3]:
```
sns.set_style('white')
```

[4]:
```
path = Path('time_gan')
hdf_store = path / 'TimeSeriesGAN.h5'
```

[5]:
```
seq_len = 24
n_seq = 6
```

2 Load Data

[6]:
```
experiment = 0
```

[7]:
```
def get_real_data():
    df = pd.read_hdf(hdf_store, 'data/real').sort_index()

    # Preprocess the dataset:
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(df)

    data = []
    for i in range(len(df) - seq_len):
        data.append(scaled_data[i:i + seq_len])
    return data


real_data = get_real_data()
```

[8]:
```
n = len(real_data)
```

[9]:
```
np.asarray(real_data).shape
```

[9]:
```
(4562, 24, 6)
```

[10]:
```
synthetic_data = np.load(path / f'experiment_{experiment:02d}' /␣
 ↪'generated_data.npy')
synthetic_data.shape
```

[10]:
```
(4480, 24, 6)
```

[11]:
```
real_data = real_data[:synthetic_data.shape[0]]
```

3 Prepare Sample

[12]:
```
sample_size = 250
idx = np.random.permutation(len(real_data))[:sample_size]
```

[13]:
```
# Data preprocessing
real_sample = np.asarray(real_data)[idx]
synthetic_sample = np.asarray(synthetic_data)[idx]
```

[14]:
```
real_sample_2d = real_sample.reshape(-1, seq_len)
synthetic_sample_2d = synthetic_sample.reshape(-1, seq_len)
```

[15]:
```
real_sample_2d.shape, synthetic_sample_2d.shape
```

[15]: ((1500, 24), (1500, 24))

4 Visualization in 2D: A Qualitative Assessment of Diversity

4.1 Run PCA

[16]:
```
pca = PCA(n_components=2)
pca.fit(real_sample_2d)
pca_real = (pd.DataFrame(pca.transform(real_sample_2d))
            .assign(Data='Real'))
pca_synthetic = (pd.DataFrame(pca.transform(synthetic_sample_2d))
                 .assign(Data='Synthetic'))
pca_result = pca_real.append(pca_synthetic).rename(
    columns={0: '1st Component', 1: '2nd Component'})
```

4.2 Run t-SNE

[17]:
```
tsne_data = np.concatenate((real_sample_2d,
                            synthetic_sample_2d), axis=0)

tsne = TSNE(n_components=2,
            verbose=1,
            perplexity=40)
tsne_result = tsne.fit_transform(tsne_data)
```

```
[t-SNE] Computing 121 nearest neighbors…
[t-SNE] Indexed 3000 samples in 0.000s…
[t-SNE] Computed neighbors for 3000 samples in 0.311s…
[t-SNE] Computed conditional probabilities for sample 1000 / 3000
[t-SNE] Computed conditional probabilities for sample 2000 / 3000
[t-SNE] Computed conditional probabilities for sample 3000 / 3000
[t-SNE] Mean sigma: 0.065877
[t-SNE] KL divergence after 250 iterations with early exaggeration: 53.936047
[t-SNE] KL divergence after 1000 iterations: 0.279739
```

[18]:
```
tsne_result = pd.DataFrame(tsne_result, columns=['X', 'Y']).assign(Data='Real')
tsne_result.loc[sample_size*6:, 'Data'] = 'Synthetic'
```

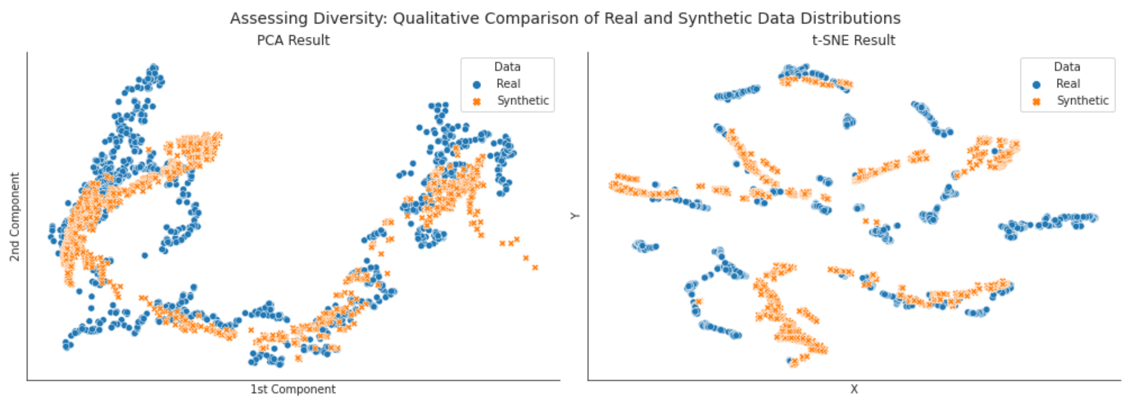
4.3 Plot Result

[19]:
```
fig, axes = plt.subplots(ncols=2, figsize=(14, 5))

sns.scatterplot(x='1st Component', y='2nd Component', data=pca_result,
                hue='Data', style='Data', ax=axes[0])
sns.despine()
axes[0].set_title('PCA Result')


sns.scatterplot(x='X', y='Y',
                data=tsne_result,
                hue='Data',
                style='Data',
                ax=axes[1])
sns.despine()
for i in [0, 1]:
    axes[i].set_xticks([])
    axes[i].set_yticks([])

axes[1].set_title('t-SNE Result')
fig.suptitle('Assessing Diversity: Qualitative Comparison of Real and Synthetic␣
↪Data Distributions',
             fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=.88);
```



5 Time Series Classification: A quantitative Assessment of Fidelity

5.1 Prepare Data

[20]:
```
real_data = get_real_data()
real_data = np.array(real_data)[:len(synthetic_data)]
real_data.shape
```

[20]: (4480, 24, 6)

[21]:
```
synthetic_data.shape
```

[21]: (4480, 24, 6)

[22]:
```
n_series = real_data.shape[0]
```

[23]:
```
idx = np.arange(n_series)
```

[24]:
```
n_train = int(.8*n_series)
train_idx = idx[:n_train]
test_idx = idx[n_train:]
```

[25]:
```
train_data = np.vstack((real_data[train_idx],
                        synthetic_data[train_idx]))
test_data = np.vstack((real_data[test_idx],
                       synthetic_data[test_idx]))
```

[26]:
```
n_train, n_test = len(train_idx), len(test_idx)
train_labels = np.concatenate((np.ones(n_train),
                               np.zeros(n_train)))
test_labels = np.concatenate((np.ones(n_test),
                              np.zeros(n_test)))
```

5.2 Create Classifier

[27]:
```
ts_classifier = Sequential([GRU(6, input_shape=(24, 6), name='GRU'),
                            Dense(1, activation='sigmoid', name='OUT')],
                           name='Time_Series_Classifier')
```

[28]:
```
ts_classifier.compile(loss='binary_crossentropy',
                      optimizer='adam',
                      metrics=[AUC(name='AUC'), 'accuracy'])
```

[29]:
```
ts_classifier.summary()
```

```
Model: "Time_Series_Classifier"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
GRU (GRU)                    (None, 6)                 252
_________________________________________________________________
OUT (Dense)                  (None, 1)                 7
=================================================================
Total params: 259
Trainable params: 259
Non-trainable params: 0
_________________________________________________________________
```

[30]:
```
result = ts_classifier.fit(x=train_data,
                           y=train_labels,
                           validation_data=(test_data, test_labels),
                           epochs=250,
                           batch_size=128,
                           verbose=0)
```

[31]:
```
ts_classifier.evaluate(x=test_data, y=test_labels)
```

```
56/56 [==============================] - 0s 2ms/step - loss: 3.3849 - AUC:
0.2817 - accuracy: 0.5307
```

[31]:
```
[3.3848929405212402, 0.2817339301109314, 0.5306919813156128]
```

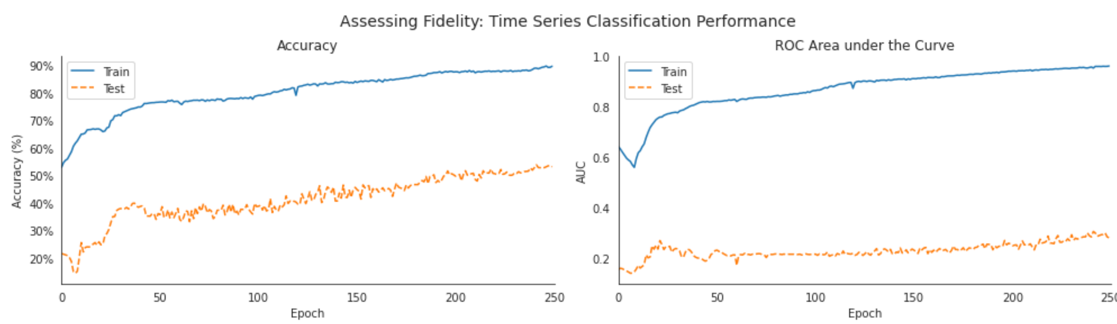
[32]:
```
history = pd.DataFrame(result.history)
history.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   loss          250 non-null    float64
 1   AUC           250 non-null    float64
 2   accuracy      250 non-null    float64
 3   val_loss      250 non-null    float64
 4   val_AUC       250 non-null    float64
 5   val_accuracy  250 non-null    float64
dtypes: float64(6)
memory usage: 11.8 KB
```

[33]:
```
from matplotlib.ticker import FuncFormatter
```

```
[34]: sns.set_style('white')
fig, axes = plt.subplots(ncols=2, figsize=(14,4))
history[['AUC', 'val_AUC']].rename(columns={'AUC': 'Train', 'val_AUC': 'Test'}).plot(ax=axes[1],
      title='ROC Area under the Curve',
     style=['-', '--'],
     xlim=(0, 250))
history[['accuracy', 'val_accuracy']].rename(columns={'accuracy': 'Train', 'val_accuracy': 'Test'}).plot(ax=axes[0],
                           title='Accuracy',
                          style=['-', '--'],
                          xlim=(0, 250))
for i in [0, 1]:
    axes[i].set_xlabel('Epoch')

axes[0].yaxis.set_major_formatter(FuncFormatter(lambda y, _: '{:.0%}'.format(y)))
axes[0].set_ylabel('Accuracy (%)')
axes[1].set_ylabel('AUC')
sns.despine()
fig.suptitle('Assessing Fidelity: Time Series Classification Performance', fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=.85);
```



6 Train on Synthetic, test on real: Assessing usefulness

[35]:
```python
real_data = get_real_data()
real_data = np.array(real_data)[:len(synthetic_data)]
```

[36]:
```python
real_data.shape, synthetic_data.shape
```

[36]: ((4480, 24, 6), (4480, 24, 6))

[37]:
```python
real_train_data = real_data[train_idx, :23, :]
real_train_label = real_data[train_idx, -1, :]

real_test_data = real_data[test_idx, :23, :]
real_test_label = real_data[test_idx, -1, :]
```

[38]:
```python
real_train_data.shape, real_train_label.shape, real_test_data.shape,␣
↪real_test_label.shape
```

[38]: ((3584, 23, 6), (3584, 6), (896, 23, 6), (896, 6))

[39]:
```python
synthetic_train = synthetic_data[:, :23, :]
synthetic_label = synthetic_data[:, -1, :]
```

[40]:
```python
synthetic_train.shape, synthetic_label.shape
```

[40]: ((4480, 23, 6), (4480, 6))

[41]:
```python
def get_model():
    model = Sequential([GRU(12, input_shape=(seq_len-1, n_seq)),
                        Dense(6)])

    model.compile(optimizer=Adam(),
                  loss=MeanAbsoluteError(name='MAE'))
    return model
```

[42]:
```python
ts_regression = get_model()
synthetic_result = ts_regression.fit(x=synthetic_train,
                                     y=synthetic_label,
                                     validation_data=(
                                         real_test_data,
                                         real_test_label),
                                     epochs=100,
                                     batch_size=128,
                                     verbose=0)
```

[43]:
```python
ts_regression = get_model()
real_result = ts_regression.fit(x=real_train_data,
```

```
                                y=real_train_label,
                                validation_data=(
                                    real_test_data,
                                    real_test_label),
                                epochs=100,
                                batch_size=128,
                                verbose=0)
```

[44]:
```
synthetic_result = pd.DataFrame(synthetic_result.history).
 ↪rename(columns={'loss': 'Train', 'val_loss': 'Test'})
real_result = pd.DataFrame(real_result.history).rename(columns={'loss':␣
 ↪'Train', 'val_loss': 'Test'})
```

[45]:
```
fig, axes = plt.subplots(ncols=2, figsize=(14, 4), sharey=True)
synthetic_result.plot(ax=axes[0], title='Train on Synthetic, Test on Real',␣
 ↪logy=True, xlim=(0, 100))
real_result.plot(ax=axes[1], title='Train on Real, Test on Real', logy=True,␣
 ↪xlim=(0, 100))
for i in [0, 1]:
    axes[i].set_xlabel('Epoch')
    axes[i].set_ylabel('Mean Absolute Error (log scale)')

sns.despine()
fig.suptitle('Assessing Usefulness: Time Series Prediction Performance',␣
 ↪fontsize=14)
fig.tight_layout()
fig.subplots_adjust(top=.85);
```