fase4_1

May 21, 2025

[8]:
```
import pandas as pd

df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv")      # Dados reais de treino

df_real.describe()
```

[8]:

	WAP001	WAP002	WAP003	WAP004	WAP005	WAP006	WAP007	\
count	1396.0	1396.0	1396.0	1396.0	1396.0	1396.000000	1396.0	
mean	-110.0	-110.0	-110.0	-110.0	-110.0	-109.584527	-110.0	
std	0.0	0.0	0.0	0.0	0.0	2.912958	0.0	
min	-110.0	-110.0	-110.0	-110.0	-110.0	-110.000000	-110.0	
25%	-110.0	-110.0	-110.0	-110.0	-110.0	-110.000000	-110.0	
50%	-110.0	-110.0	-110.0	-110.0	-110.0	-110.000000	-110.0	
75%	-110.0	-110.0	-110.0	-110.0	-110.0	-110.000000	-110.0	
max	-110.0	-110.0	-110.0	-110.0	-110.0	-88.000000	-110.0	

	WAP008	WAP009	WAP010	…	WAP520	LONGITUDE	\
count	1396.000000	1396.0	1396.000000	…	1396.0	1396.000000	
mean	-109.197708	-110.0	-109.678367	…	-110.0	-7486.581784	
std	3.946364	0.0	2.181149	…	0.0	45.101037	
min	-110.000000	-110.0	-110.000000	…	-110.0	-7571.093400	
25%	-110.000000	-110.0	-110.000000	…	-110.0	-7520.755800	
50%	-110.000000	-110.0	-110.000000	…	-110.0	-7491.030634	
75%	-110.000000	-110.0	-110.000000	…	-110.0	-7443.877677	
max	-80.000000	-110.0	-92.000000	…	-110.0	-7408.695251	

	LATITUDE	FLOOR	BUILDINGID	SPACEID	RELATIVEPOSITION	\
count	1.396000e+03	1396.0	1396.0	1396.000000	1396.000000	
mean	4.864879e+06	2.0	1.0	117.111748	1.704155	
std	3.501884e+01	0.0	0.0	83.279968	0.456585	
min	4.864810e+06	2.0	1.0	2.000000	1.000000	
25%	4.864859e+06	2.0	1.0	17.000000	1.000000	
50%	4.864873e+06	2.0	1.0	107.000000	2.000000	
75%	4.864893e+06	2.0	1.0	204.000000	2.000000	
max	4.864959e+06	2.0	1.0	217.000000	2.000000	

```
            USERID      PHONEID     TIMESTAMP
count  1396.000000  1396.000000  1.396000e+03
mean      5.461318    17.108883  1.371721e+09
std       3.304272     5.297423  9.536837e+03
min       2.000000     8.000000  1.371714e+09
25%       2.000000    14.000000  1.371714e+09
50%       4.000000    18.000000  1.371715e+09
75%       9.000000    23.000000  1.371735e+09
max      10.000000    23.000000  1.371738e+09

[8 rows x 529 columns]
```

[9]:
```
df_real.columns
```

[9]:
```
Index(['WAP001', 'WAP002', 'WAP003', 'WAP004', 'WAP005', 'WAP006', 'WAP007',
       'WAP008', 'WAP009', 'WAP010',
       …
       'WAP520', 'LONGITUDE', 'LATITUDE', 'FLOOR', 'BUILDINGID', 'SPACEID',
       'RELATIVEPOSITION', 'USERID', 'PHONEID', 'TIMESTAMP'],
      dtype='object', length=529)
```

[11]:
```
df_generated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_generated_pseudo.csv") # Vetores pseudo-rotulados gerados
df_generated.describe()
```

[11]:
```
             WAP001        WAP002        WAP003        WAP004        WAP005  \
count  40000.000000  40000.000000  40000.000000  40000.000000  40000.000000
mean     -52.131000    -97.930100    -60.247600    -89.294925    -63.500625
std        4.353879      5.075836      4.681441      4.961357      3.109902
min      -64.000000   -110.000000    -71.000000   -110.000000    -72.000000
25%      -55.000000   -101.000000    -64.000000    -93.000000    -66.000000
50%      -53.000000    -98.000000    -61.000000    -89.000000    -64.000000
75%      -49.000000    -94.000000    -57.000000    -86.000000    -62.000000
max      -40.000000    -82.000000    -40.000000    -78.000000    -46.000000

             WAP006        WAP007        WAP008        WAP009        WAP010  \
count  40000.000000  40000.000000  40000.000000  40000.000000  40000.000000
mean     -94.420700    -70.493825    -97.648175    -93.546025    -67.002625
std        5.060805      2.509307      5.459940      3.898392      4.002002
min     -110.000000    -79.000000   -110.000000   -110.000000    -78.000000
25%      -98.000000    -72.000000   -101.000000    -96.000000    -70.000000
50%      -94.000000    -71.000000    -97.000000    -93.000000    -67.000000
75%      -91.000000    -69.000000    -93.000000    -91.000000    -64.000000
max      -80.000000    -60.000000    -85.000000    -82.000000    -50.000000

          LONGITUDE      LATITUDE
count  40000.000000  40000.000000
```

```
mean      10.037707      9.903981
std        1.724494      1.243719
min        3.990654      6.160164
25%        8.866893      8.976322
50%       10.104205      9.870904
75%       11.266562     10.831815
max       15.545594     13.339489
```

[12]:
```
df_generated.columns
```

[12]:
```
Index(['WAP001', 'WAP002', 'WAP003', 'WAP004', 'WAP005', 'WAP006', 'WAP007',
       'WAP008', 'WAP009', 'WAP010', 'LONGITUDE', 'LATITUDE'],
      dtype='object')
```

[15]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Phase 4: Reproduce Table 2 and Figure 6
# ---------------------------------------
# 1. Load datasets
# ---------------------------------------
df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv")      # Dados reais de treino
df_generated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_generated_pseudo.csv") # Vetores pseudo-rotulados gerados
df_test = pd.read_csv('/home/darkcover/Documentos/Gan/Data/df_test.csv') #␣
 ↪Conjunto de teste fixo

# 2. Determine WAP columns common to all datasets
# ---------------------------------------
wap_real = {col for col in df_real.columns if col.startswith('WAP')}
wap_gen  = {col for col in df_generated.columns if col.startswith('WAP')}
wap_test = {col for col in df_test.columns if col.startswith('WAP')}
# Use only the WAPs present in real, generated, and test
wap_columns = sorted(list(wap_real & wap_gen & wap_test))

# 2.1 Prepare features (X) and labels (y)
# ---------------------------------------
X_real      = df_real[wap_columns].values.astype(np.float32)
y_real      = df_real[['LONGITUDE','LATITUDE']].values.astype(np.float32)

X_gen_full  = df_generated[wap_columns].values.astype(np.float32)
y_gen_full  = df_generated[['LONGITUDE','LATITUDE']].values.astype(np.float32)

X_test      = df_test[wap_columns].values.astype(np.float32)
```

```
y_test      = df_test[['X','Y']].values.astype(np.float32)

# 3. Model builder
def build_model(input_dim, learning_rate):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(input_dim,)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(2)
    ])
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer,
                  loss='mse')
    return model

# 4. Training and evaluation function
# ------------------------------------

def train_and_evaluate(X_train, y_train, X_eval, y_eval, config,␣
 ↪return_errors=False):
    """
    Treina uma DNN e avalia o erro de localização.
    config: dict com keys 'epochs', 'batch_size', 'learning_rate'
    return_errors: se True, retorna array de erros ponta a ponta.
    """
    model = build_model(input_dim=X_train.shape[1],␣
 ↪learning_rate=config['learning_rate'])
    model.fit(X_train, y_train,
              epochs=config['epochs'],
              batch_size=config['batch_size'],
              verbose=0)
    y_pred = model.predict(X_eval)
    # erro Euclidiano por ponto
    errs = np.linalg.norm(y_pred - y_eval, axis=1)
    if return_errors:
        return errs
    return errs.mean()

# 5. Configurações de experimento (conforme artigo)
# ------------------------------------
configs = {
    'Supervised(1000,100)':   {'epochs': 250, 'batch_size': 100,␣
 ↪'learning_rate': 0.01},
    'Supervised(2000,1100)': {'epochs': 250, 'batch_size': 100, 'learning_rate':
 ↪ 0.01}
}
# Adicionando configurações Selective-SS-GAN para diferentes ms
for ms in [100, 500, 1000]:
```

```
    key = f'Selective-SS-GAN(1000,100,{ms})'
    configs[key] = {'epochs': 250, 'batch_size': 100, 'learning_rate': 0.01}

# 6. Executar experimentos
# ------------------------------------
results = {}

# Baseline 1: Supervised(1000,100)
X_train = X_real[:1000]
y_train = y_real[:1000]
results['Supervised(1000,100)'] = train_and_evaluate(
    X_train, y_train, X_test, y_test, configs['Supervised(1000,100)']
)

# Baseline 2: Supervised(2000,1100) duplicando reais
X2 = np.vstack([X_train, X_train])
y2 = np.vstack([y_train, y_train])
results['Supervised(2000,1100)'] = train_and_evaluate(
    X2, y2, X_test, y_test, configs['Supervised(2000,1100)']
)

# Selective-SS-GAN experiments
for ms in [100, 500, 1000]:
    key = f'Selective-SS-GAN(1000,100,{ms})'
    X_mix = np.vstack([X_train, X_gen_full[:ms]])
    y_mix = np.vstack([y_train, y_gen_full[:ms]])
    results[key] = train_and_evaluate(
        X_mix, y_mix, X_test, y_test, configs[key]
    )

# 7. Reproduzir Tabela 2
# ------------------------------------
table2 = pd.DataFrame(
    [(method, err) for method, err in results.items()],
    columns=['Method', 'MeanError']
)
print("Tabela 2:\n", table2)

# 8. Gerar Figura 6 (CDF de erros)
# ------------------------------------
errs_base = train_and_evaluate(X_train, y_train, X_test, y_test,
                               configs['Supervised(1000,100)'],
                               return_errors=True)
errs_select = train_and_evaluate(
    np.vstack([X_train, X_gen_full[:1000]]),
    np.vstack([y_train, y_gen_full[:1000]]),
    X_test, y_test,
```

```
    configs['Selective-SS-GAN(1000,100,1000)'],
    return_errors=True
)

plt.figure(figsize=(6,4))
for errs, label in [(errs_base, 'Supervised'), (errs_select,␣
 ↪'Selective-SS-GAN')]:
    sorted_e = np.sort(errs)
    cdf = np.arange(len(sorted_e)) / float(len(sorted_e))
    plt.plot(sorted_e, cdf, label=label)

plt.xlabel('Localization error (m)')
plt.ylabel('CDF')
plt.title('Figure 6. Error CDF comparison')
plt.legend()
plt.grid(True, linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.show()
```

```
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
25/25             0s 4ms/step
25/25             0s 3ms/step
25/25             0s 3ms/step
25/25             0s 2ms/step
25/25             0s 3ms/step
Tabela 2:
                            Method   MeanError
0             Supervised(1000,100)  3903426.25
1            Supervised(2000,1100)  3899272.25
2   Selective-SS-GAN(1000,100,100)  3806442.25
3   Selective-SS-GAN(1000,100,500)  3830636.25
4  Selective-SS-GAN(1000,100,1000)  3849217.25
```

```
/home/darkcover/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-
py3.12/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
25/25             0s 4ms/step
25/25             0s 2ms/step
```

Figure 6. Error CDF comparison