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')
```

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

# 1. Carregar datasets
df_real = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv")
df_generated = pd.read_csv("/home/darkcover/Documentos/Gan/Data/
 ↪df_generated_pseudo.csv")
df_test = pd.read_csv("/home/darkcover/Documentos/Gan/Data/df_test.csv")

# 2. Selecionar colunas WAP comuns
wap_real = {c for c in df_real.columns if c.startswith('WAP')}
wap_gen  = {c for c in df_generated.columns if c.startswith('WAP')}
wap_test = {c for c in df_test.columns      if c.startswith('WAP')}
wap_columns = sorted(list(wap_real & wap_gen & wap_test))

# 2.1 Preparar X e 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. Funções de modelo e treino
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)
```

```
    ])
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate), loss='mse')
    return model

def train_and_evaluate(X_train, y_train, X_eval, y_eval, cfg,␣
 ↪return_errors=False):
    model = build_model(X_train.shape[1], cfg['learning_rate'])
    model.fit(X_train, y_train,
              epochs=cfg['epochs'],
              batch_size=cfg['batch_size'],
              verbose=0)
    y_pred = model.predict(X_eval)
    errs = np.linalg.norm(y_pred - y_eval, axis=1)
    return errs if return_errors else errs.mean()

# 4. Configurações de experimento
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},
}
for ms in [100, 500, 1000]:
    configs[f'Selective-SS-GAN(1000,100,{ms})'] = {
        'epochs':250, 'batch_size':100, 'learning_rate':0.01
    }

# 5. Executar experimentos e coletar erros
# 5.1 Supervised(1000,100)
X_train    = X_real[:1000]
y_train    = y_real[:1000]
errs_sup100   = train_and_evaluate(X_train, y_train, X_test, y_test,
                                   configs['Supervised(1000,100)'],
                                   return_errors=True)

# 5.2 Supervised(2000,1100) duplicando amostras
X2 = np.vstack([X_train, X_train])
y2 = np.vstack([y_train, y_train])
errs_sup2000 = train_and_evaluate(X2, y2, X_test, y_test,
                                  configs['Supervised(2000,1100)'],
                                  return_errors=True)

# 5.3 Selective-SS-GAN
errs_sel = {}
for ms in [100, 500, 1000]:
    key  = f'Selective-SS-GAN(1000,100,{ms})'
    X_mix = np.vstack([X_train, X_gen_full[:ms]])
```

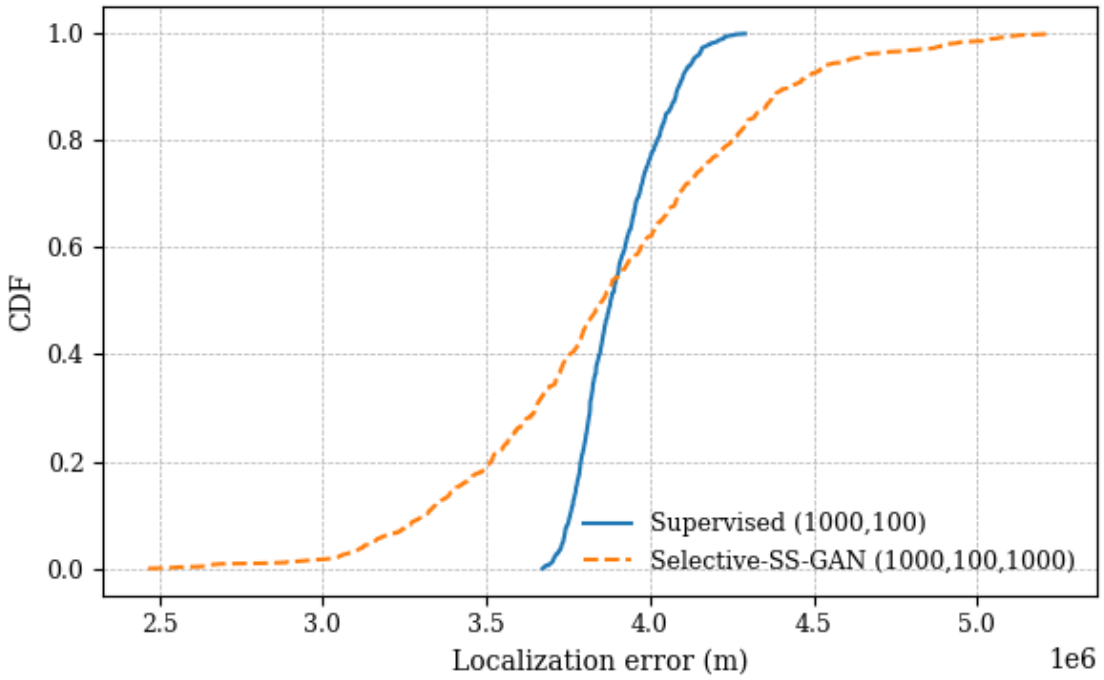
```
    y_mix = np.vstack([y_train, y_gen_full[:ms]])
    errs_sel[ms] = train_and_evaluate(X_mix, y_mix, X_test, y_test,
                                      configs[key],
                                      return_errors=True)
```

```
/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 3ms/step
25/25             0s 3ms/step
25/25             0s 2ms/step
25/25             0s 2ms/step
25/25             0s 2ms/step
```

```
<pandas.io.formats.style.Styler at 0x79689a86f320>
```

Fig. 6. Localization performance comparison



[ ]:
```
# 6. Reproduzir Tabela 2 (média, mínimo e máximo)
data = []
data.append({
    'Método': 'Supervised(1000,100)',
    'Erro médio (m)':  np.mean(errs_sup100),
```

```
    'Erro mínimo (m)': np.min(errs_sup100),
    'Erro máximo (m)': np.max(errs_sup100),
})
data.append({
    'Método': 'Supervised(2000,1100)',
    'Erro médio (m)':  np.mean(errs_sup2000),
    'Erro mínimo (m)': np.min(errs_sup2000),
    'Erro máximo (m)': np.max(errs_sup2000),
})
for ms in [100, 500, 1000]:
    lbl = f'Selective-SS-GAN(1000,100,{ms})'
    errs = errs_sel[ms]
    data.append({
        'Método':           lbl,
        'Erro médio (m)':   np.mean(errs),
        'Erro mínimo (m)':  np.min(errs),
        'Erro máximo (m)':  np.max(errs),
    })

df_table2 = pd.DataFrame(data)
styled = (
    df_table2.style
        .hide(axis='index')               # esconde o índice
        .set_caption('Tabela 2. Performance de localização considerando 1000␣
 ↪amostras rotuladas.')
        .format({
            'Erro médio (m)': '{:.3f}',
            'Erro mínimo (m)': '{:.3f}',
            'Erro máximo (m)': '{:.3f}',
        })
        .set_table_styles([
            {'selector': 'caption',
             'props': [('caption-side','bottom'),
                       ('font-weight','bold'),
                       ('text-align','center')]},
            {'selector': 'th',
             'props': [('font-weight','bold'),
                       ('text-align','center')]},
            {'selector': 'td',
             'props': [('text-align','center')]},
        ])
)
display(styled)


# 7. Gerar Figura 6 (CDF de erros)
# 7.1 Preparar CDF
```

```
sorted_base   = np.sort(errs_sup100)
cdf_base      = np.arange(len(sorted_base)) / float(len(sorted_base))
sorted_sel1000 = np.sort(errs_sel[1000])
cdf_sel1000    = np.arange(len(sorted_sel1000)) / float(len(sorted_sel1000))

# 7.2 Plotagem
plt.rc('font', family='serif', size=10)
plt.figure(figsize=(6,4), dpi=100)
plt.plot(sorted_base,    cdf_base,    linestyle='-',  linewidth=1.5,
         label='Supervised (1000,100)')
plt.plot(sorted_sel1000, cdf_sel1000, linestyle='--', linewidth=1.5,
         label='Selective-SS-GAN (1000,100,1000)')
plt.xlabel('Localization error (m)', fontsize=10)
plt.ylabel('CDF',                fontsize=10)
plt.title('Fig. 6. Localization performance comparison',
          fontsize=11, fontweight='bold')
plt.xticks(fontsize=9)
plt.yticks(fontsize=9)
plt.grid(True, linestyle='--', linewidth=0.5)
plt.legend(frameon=False, loc='lower right', fontsize=9)
plt.tight_layout()
plt.show()
```