fase4_1

May 18, 2025

[6]:
```
# Avaliacao com base UJIIndoorLoc (completa)
# Etapas: leitura, limpeza, treino com e sem GAN, Tabela 2, Figura 6, Tabela 3

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

# ============================
# 1. Carregar base UJIIndoorLoc
# ============================
file_path = "/home/darkcover/Documentos/Gan/Data/
 ↪ujindoorsubset_building1_floor2.csv"
df = pd.read_csv(file_path)

# ============================
# 2. Selecionar colunas WAP válidas
# ============================
wap_cols = [col for col in df.columns if col.startswith("WAP")]
df_wap = df[wap_cols].copy()
df_wap = df_wap.loc[:, (df_wap != -110).any(axis=0)]

# Top 10 WAPs mais informativos
missing_ratio = (df_wap == -110).sum() / len(df_wap)
selected_waps = missing_ratio.sort_values().head(10).index.tolist()
print("  WAPs Selecionados:", selected_waps)

# ============================
# 3. Selecionar amostras com >= 6 RSSI válidos
# ============================
df_wap_selected = df[selected_waps].replace(-110, np.nan)
valid_rows = df_wap_selected.notna().sum(axis=1) >= 6
```

```
# Combinar com coordenadas e limpar
df_selected = df.loc[valid_rows, selected_waps + ['LONGITUDE', 'LATITUDE']].
 ↪copy()
df_selected = df_selected.replace(-110, np.nan).dropna().reset_index(drop=True)

X = df_selected[selected_waps].values.astype(np.float32)
y = df_selected[['LONGITUDE', 'LATITUDE']].values.astype(np.float32)

# ============================
# 4. Dividir em treino e teste
# ============================
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
print(f" Base UJI pronta | Treino: {X_train.shape[0]} | Teste: {X_test.
 ↪shape[0]}")

# ============================
# 5. Carregar vetores gerados + pseudo-label (df_generated)
# ============================
df_gen = pd.read_csv("Data/df_generated.csv")
X_gen = df_gen[selected_waps].values.astype(np.float32)
y_gen = df_gen[['LONGITUDE', 'LATITUDE']].values.astype(np.float32)

# ============================
# 6. Treinar dois modelos (real vs real+gerado)
# ============================
def build_model():
    model = Sequential([
        Dense(64, activation='relu', input_shape=(10,)),
        Dense(64, activation='relu'),
        Dense(2)
    ])
    model.compile(optimizer=Adam(0.001), loss='mse')
    return model

# Modelo A: apenas dados reais
model_real = build_model()
model_real.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)

# Modelo B: reais + gerados
X_aug = np.vstack([X_train, X_gen])
y_aug = np.vstack([y_train, y_gen])
model_aug = build_model()
model_aug.fit(X_aug, y_aug, epochs=150, batch_size=32, verbose=0)

# ============================
# 7. Avaliar modelos
```

```
# ============================
pred_real = model_real.predict(X_test)
pred_aug = model_aug.predict(X_test)

error_real = np.linalg.norm(pred_real - y_test, axis=1)
error_aug = np.linalg.norm(pred_aug - y_test, axis=1)

print("\n Tabela 2 - Erro médio:")
print(f"Reais:     MAE = {np.mean(error_real):.3f} m")
print(f"Reais+GAN: MAE = {np.mean(error_aug):.3f} m")

# ============================
# 8. Figura 6 - Histograma de erros
# ============================
plt.figure(figsize=(8,5))
plt.hist(error_real, bins=30, alpha=0.6, label='Real only')
plt.hist(error_aug, bins=30, alpha=0.6, label='Real + Generated')
plt.xlabel("Localization error (m)")
plt.ylabel("Samples")
plt.title("FIGURE 6 - Localization error histogram")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ============================
# 9. Tabela 3 - Cobertura de zonas
# ============================
def zone_coverage(df, label):
    df['zone_x'] = df['LONGITUDE'].astype(int)
    df['zone_y'] = df['LATITUDE'].astype(int)
    df['zone'] = df['zone_x'].astype(str) + '_' + df['zone_y'].astype(str)
    n_zones = df['zone'].nunique()
    print(f"{label}: Cobertura = {n_zones} zonas")

print("\n Tabela 3 - Cobertura de Zonas")
zone_coverage(df_selected, "Apenas reais")
zone_coverage(pd.concat([df_selected, df_gen], ignore_index=True), "Reais +␣
 ↪Gerados")
```

```
 WAPs Selecionados: ['WAP179', 'WAP016', 'WAP015', 'WAP178', 'WAP189',
'WAP116', 'WAP115', 'WAP114', 'WAP223', 'WAP188']
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 47
     42 y = df_selected[['LONGITUDE', 'LATITUDE']].values.astype(np.float32)
```

```
     44 # ============================
     45 # 4. Dividir em treino e teste
     46 # ============================
---> 47 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
     48 print(f"  Base UJI pronta | Treino: {X_train.shape[0]} | Teste: {X_test.shape[0]}")
     50 # ============================
     51 # 5. Carregar vetores gerados + pseudo-label (df_generated)
     52 # ============================

File ~/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-py3.12/lib/python3.12/site-packages/sklearn/utils/_param_validation.py:216, in validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    210 try:
    211     with config_context(
    212         skip_parameter_validation=(
    213             prefer_skip_nested_validation or global_skip_validation
    214         )
    215     ):
--> 216         return func(*args, **kwargs)
    217 except InvalidParameterError as e:
    218     # When the function is just a wrapper around an estimator, we allow
    219     # the function to delegate validation to the estimator, but we replace
    220     # the name of the estimator by the name of the function in the erro
    221     # message to avoid confusion.
    222     msg = re.sub(
    223         r"parameter of \w+ must be",
    224         f"parameter of {func.__qualname__} must be",
    225         str(e),
    226     )

File ~/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-py3.12/lib/python3.12/site-packages/sklearn/model_selection/_split.py:2851, in train_test_split(test_size, train_size, random_state, shuffle, stratify, *arrays)
  2848 arrays = indexable(*arrays)
  2850 n_samples = _num_samples(arrays[0])
-> 2851 n_train, n_test = _validate_shuffle_split(
  2852     n_samples, test_size, train_size, default_test_size=0.25
  2853 )
  2855 if shuffle is False:
  2856     if stratify is not None:

File ~/.cache/pypoetry/virtualenvs/gan-oPyfrVEv-py3.12/lib/python3.12/site-packages/sklearn/model_selection/_split.py:2481, in _validate_shuffle_split(n_samples, test_size, train_size, default_test_size)
  2478 n_train, n_test = int(n_train), int(n_test)
```

```
   2480 if n_train == 0:
-> 2481     raise ValueError(
   2482         "With n_samples={}, test_size={} and train_size={}, the "
   2483         "resulting train set will be empty. Adjust any of the "
   2484         "aforementioned parameters.".format(n_samples, test_size,␣
 ↪train_size)
   2485     )
   2487 return n_train, n_test

ValueError: With n_samples=0, test_size=0.3 and train_size=None, the resulting␣
 ↪train set will be empty. Adjust any of the aforementioned parameters.
```