artigo

June 28, 2025

[ ]:
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Criar pastas de saída
os.makedirs('processed', exist_ok=True)
os.makedirs('plots', exist_ok=True)

# 1. Fetch dataset Metro Interstate Traffic Volume (id=492)
print('Carregando dataset...')
dataset = fetch_ucirepo(id=492)
df = dataset.data.features.copy()
df['traffic_volume'] = dataset.data.targets.astype(int)

df['date_time'] = pd.to_datetime(df['date_time'])
df.sort_values('date_time', inplace=True)
df.reset_index(drop=True, inplace=True)

# 2. Salvar CSV completo
df.to_csv('./processed/traffic_full.csv', index=False)
print('CSV completo salvo em processed/traffic_full.csv')

# 3. EDA: plot série completa
plt.figure(figsize=(12,4))
plt.plot(df['date_time'], df['traffic_volume'])
plt.title('Traffic Volume Over Time')
plt.xlabel('Date Time')
plt.ylabel('Volume')
plt.tight_layout()
plt.savefig('./plots/traffic_full.png')
plt.close()
print('Plot completo salvo em plots/traffic_full.png')
```

```python
# 4. Plot por ano
for year, grp in df.groupby(df['date_time'].dt.year):
    plt.figure(figsize=(12,4))
    plt.plot(grp['date_time'], grp['traffic_volume'])
    plt.title(f'Traffic Volume in {year}')
    plt.xlabel('Date Time')
    plt.ylabel('Volume')
    plt.tight_layout()
    path = f'./plots/traffic_{year}.png'
    plt.savefig(path)
    plt.close()
    print(f'Plot {year} salvo em {path}')

# 4b. Plot por mês (ano-mês)
for (year, month), grp in df.groupby([df['date_time'].dt.year, df['date_time'].dt.month]):
    plt.figure(figsize=(12,4))
    plt.plot(grp['date_time'], grp['traffic_volume'])
    plt.title(f'Traffic Volume in {year}-{month:02d}')
    plt.xlabel('Date Time')
    plt.ylabel('Volume')
    plt.tight_layout()
    path = f'./plots/traffic_{year}_{month:02d}.png'
    plt.savefig(path)
    plt.close()
    print(f'Plot {year}-{month:02d} salvo em {path}')

# 5. Anomalias via Z-score univariada (threshold=2)
traffic = df['traffic_volume'].values
z_scores = np.abs(stats.zscore(traffic))
thresh = 2
anomaly_df = df[z_scores > thresh].copy()
anomaly_df.to_csv('./processed/anomalies_zscore.csv', index=False)
print('Anomalias Z-score salvas em processed/anomalies_zscore.csv')

# 6. Pré-processamento multivariado
features = ['holiday','temp','rain_1h','snow_1h','clouds_all','weather_main']
preprocessor = ColumnTransformer([
    ('cat', OneHotEncoder(drop='first', sparse_output=False), ['holiday','weather_main']),
    ('num', MinMaxScaler(), ['temp','rain_1h','snow_1h','clouds_all'])
])
X_all = preprocessor.fit_transform(df[features])
y_all = df['traffic_volume'].values

# Montar DataFrame para CSV preprocessed
```

```python
cols = preprocessor.get_feature_names_out()
df_proc = pd.DataFrame(X_all, columns=cols)
df_proc['traffic_volume'] = y_all
df_proc['date_time'] = df['date_time'].values
df_proc.to_csv('./processed/preprocessed_features.csv', index=False)
print('Recursos pré-processados salvos em processed/preprocessed_features.csv')

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Carregar dados pré-processados
df = pd.read_csv('processed/preprocessed_features.csv')
window = 24

# Extrair datas, features e target
dates = pd.to_datetime(df['date_time'])
y = df['traffic_volume'].values
X = df.drop(columns=['traffic_volume','date_time']).values

# Função para criar sequências
def create_sequences(X, y, window):
    seq_X, seq_y = [], []
    for i in range(len(X) - window):
        seq_X.append(X[i:i+window])
        seq_y.append(y[i+window])
    return np.array(seq_X), np.array(seq_y)

# Criar sequências e dividir cronologicamente
total_seq, total_y = create_sequences(X, y, window)
split_idx = int(len(total_seq) * 0.7)
X_train, y_train = total_seq[:split_idx], total_y[:split_idx]
X_test,  y_test  = total_seq[split_idx:], total_y[split_idx:]

# Converter para tensores
torch.manual_seed(42)
X_train_t = torch.from_numpy(X_train).float()
y_train_t = torch.from_numpy(y_train).float().unsqueeze(1)
X_test_t  = torch.from_numpy(X_test).float()
y_test_t  = torch.from_numpy(y_test).float().unsqueeze(1)

train_ds = TensorDataset(X_train_t, y_train_t)
```

```python
test_ds  = TensorDataset(X_test_t, y_test_t)
train_loader = DataLoader(train_ds, batch_size=32, shuffle=True)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Modelo LSTM
torch.manual_seed(42)
class LSTMAnomalyDetect(nn.Module):
    def __init__(self, n_features, n_hidden=64, n_layers=2, drop=0.2):
        super().__init__()
        self.lstm = nn.LSTM(input_size=n_features, hidden_size=n_hidden,
                            num_layers=n_layers, dropout=drop, batch_first=True)
        self.fc = nn.Linear(n_hidden, 1)
    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :]
        return self.fc(out)

model = LSTMAnomalyDetect(n_features=X_train.shape[2]).to(device)
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# Treino com early stopping
best_loss, patience, trials = np.inf, 5, 0
for epoch in range(1, 51):
    model.train()
    losses = []
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        loss = criterion(model(xb), yb)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
    val_pred = model(X_train_t.to(device)).detach().cpu().numpy().flatten()
    val_loss = mean_squared_error(y_train, val_pred)
    print(f"Epoch {epoch}: train MSE={np.mean(losses):.4f}, valid MSE={val_loss:
 ↪.4f}")
    if val_loss < best_loss:
        best_loss = val_loss; trials = 0
        torch.save(model.state_dict(), 'best_lstm.pt')
    else:
        trials += 1
        if trials >= patience:
            print("Early stopping")
            break
```

```python
# Carregar melhor modelo e avaliar
model.load_state_dict(torch.load('best_lstm.pt'))
model.eval()

# Previsões e métricas
with torch.no_grad():
    y_train_pred = model(X_train_t.to(device)).cpu().numpy().flatten()
    y_test_pred  = model(X_test_t.to(device)).cpu().numpy().flatten()

# Métricas de regressão
train_mse = mean_squared_error(y_train, y_train_pred)
train_mae = mean_absolute_error(y_train, y_train_pred)
test_mse  = mean_squared_error(y_test, y_test_pred)
test_mae  = mean_absolute_error(y_test, y_test_pred)
print(f"TRAIN -> MSE: {train_mse:.2f}, MAE: {train_mae:.2f}")
print(f" TEST -> MSE: {test_mse:.2f}, MAE: {test_mae:.2f}")

# Threshold para detecção de anomalias a partir do erro de treino
train_errors = np.abs(y_train - y_train_pred)
thresh = train_errors.mean() + 2 * train_errors.std()
print(f"Threshold de anomalia (treino): {thresh:.2f}")

# Detectar anomalias no conjunto de teste
test_errors = np.abs(y_test - y_test_pred)
idx_anom = np.where(test_errors > thresh)[0]
print(f"Anomalias detectadas no TEST: {len(idx_anom)} de {len(y_test)} pontos")

# Plot de predição vs real e anomalias
times_test = dates.iloc[window+split_idx:].reset_index(drop=True)
plt.figure(figsize=(12,4))
plt.plot(times_test, y_test, label='True')
plt.plot(times_test, y_test_pred, label='Pred')
plt.scatter(times_test.iloc[idx_anom], y_test[idx_anom], c='red', s=10,␣
 ↪label='Anomalia')
plt.title('Detecção de Anomalias (Teste)')
plt.xlabel('Date Time')
plt.ylabel('Volume')
plt.legend(); plt.tight_layout()
plt.savefig('plots/anomaly_detection_test.png')
plt.close()
print('Plot de anomalias de teste salvo em plots/anomaly_detection_test.png')

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Configurações
dir_results = 'results'
dir_plots = 'plots'
s=os.makedirs(dir_results, exist_ok=True)
os.makedirs(dir_plots, exist_ok=True)

# 1) Carregar dados brutos de tráfego
# Usamos o CSV completo para manter timestamp e volume
df = pd.read_csv('processed/traffic_full.csv')
df['date_time'] = pd.to_datetime(df['date_time'])
df['hour'] = df['date_time'].dt.hour

def split_train_test(df, train_frac=0.7):
    split_time = df['date_time'].quantile(train_frac)
    train = df[df['date_time'] <= split_time]
    test  = df[df['date_time'] >  split_time]
    return train, test

train_df, test_df = split_train_test(df)

# 2) Calcular baseline por hora (média e desvio)
hour_stats = train_df.groupby('hour')['traffic_volume'] \
    .agg(['mean','std']).reset_index().rename(columns={'mean':'hour_mean','std':
 ↪'hour_std'})

# 3) Identificar anomalias no teste
# Mesclar stats no test
test_df = test_df.merge(hour_stats, how='left', on='hour')
# Definir threshold k sigma (k=2)
test_df['threshold'] = test_df['hour_mean'] + 2 * test_df['hour_std']
# Flag de anomalia
anoms = test_df[test_df['traffic_volume'] > test_df['threshold']].copy()

# 4) Salvar resultados
train_df.to_csv(os.path.join(dir_results,'hourly_baseline_train.csv'),␣
 ↪index=False)
anoms.to_csv(os.path.join(dir_results,'anomalies_by_hour_threshold.csv'),␣
 ↪index=False)
print(f"Anomalias detectadas: {len(anoms)} de {len(test_df)} pontos no teste")

# 5) Gráfico: distribuição de anomalias por hora do dia
hour_counts = anoms['hour'].value_counts().sort_index()
plt.figure(figsize=(8,4))
sns.barplot(x=hour_counts.index, y=hour_counts.values)
plt.title('Anomalias por Hora do Dia (Threshold por Hora)')
plt.xlabel('Hora')
```

```
plt.ylabel('Número de Anomalias')
plt.tight_layout()
plt.savefig(os.path.join(dir_plots,'anomalies_by_hour_threshold.png'))
plt.close()

# 6) Timeline das anomalias
plt.figure(figsize=(12,4))
plt.plot(test_df['date_time'], test_df['traffic_volume'], alpha=0.3,␣
 ↪label='Traffic')
plt.scatter(anoms['date_time'], anoms['traffic_volume'], color='red', s=10,␣
 ↪label='Anomalias')
plt.title('Anomalias pelo Threshold por Hora - Timeline')
plt.xlabel('Date Time')
plt.ylabel('Volume')
plt.legend(); plt.tight_layout()
plt.savefig(os.path.join(dir_plots,'anomalies_timeline_hour_threshold.png'))
plt.close()

print('Análises por hora concluídas.')


import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Configurações
dir_results = 'results'
dir_plots = 'plots'
window_sizes = [7, 14, 30]  # janelas em dias
os.makedirs(dir_results, exist_ok=True)
os.makedirs(dir_plots, exist_ok=True)

# 1) Carregar dados brutos
df = pd.read_csv('processed/traffic_full.csv')
df['date_time'] = pd.to_datetime(df['date_time'])
df.sort_values('date_time', inplace=True)
df.reset_index(drop=True, inplace=True)
# Extrair hora
df['hour'] = df['date_time'].dt.hour

# 2) Para cada janela, calcular threshold móvel e gerar resultados
for window in window_sizes:
    df_copy = df.copy()
    days = window  # número de dias para rolling
```

```
    # Rolling: precisamos de tantas observações quanto dias por hora
    df_copy['rolling_mean'] = df_copy.groupby('hour')['traffic_volume'].
↪transform(lambda x: x.shift().rolling(window=days, min_periods=3).mean())
    df_copy['rolling_std'] = df_copy.groupby('hour')['traffic_volume'].
↪transform(lambda x: x.shift().rolling(window=days, min_periods=3).std())

    # Threshold e flag
    df_copy['threshold'] = df_copy['rolling_mean'] + 2 * df_copy['rolling_std']
    df_copy['is_anomaly'] = df_copy['traffic_volume'] > df_copy['threshold']
    anoms = df_copy[df_copy['is_anomaly']].copy()
    print(f"Window {window} dias: {len(anoms)} anomalias de {len(df_copy)}␣
↪pontos")

    # 3) Salvar CSV de anomalias
    fname = f'rolling_{window}d_anomalies.csv'
    anoms.to_csv(os.path.join(dir_results, fname), index=False)

    # 4) Resumo por hora
    summary = df_copy.groupby('hour').agg(
    total=('traffic_volume','size'),anomalies=('is_anomaly','sum')).
↪reset_index()
    summary['anomaly_rate'] = summary['anomalies'] / summary['total']
    sname = f'rolling_{window}d_summary.csv'
    summary.to_csv(os.path.join(dir_results, sname), index=False)

    # 5) Plot taxa de anomalias por hora
    plt.figure(figsize=(10,4))
    sns.lineplot(x='hour', y='anomaly_rate', data=summary, marker='o')
    plt.title(f'Taxa de Anomalias por Hora (Rolling {window} dias)')
    plt.xlabel('Hora')
    plt.ylabel('Taxa de Anomalias')
    plt.tight_layout()
    pname = f'rolling_{window}d_rate_by_hour.png'
    plt.savefig(os.path.join(dir_plots, pname))
    plt.close()

    # 6) Timeline completo
    plt.figure(figsize=(12,4))
    plt.plot(df_copy['date_time'], df_copy['traffic_volume'], alpha=0.3)
    plt.scatter(anoms['date_time'], anoms['traffic_volume'], color='red', s=8)
    plt.title(f'Anomalias (Rolling {window} dias)')
    plt.xlabel('Date Time')
    plt.ylabel('Volume')
    plt.tight_layout()
    tname = f'rolling_{window}d_timeline.png'
    plt.savefig(os.path.join(dir_plots, tname))
    plt.close()
```

```python
print('Análises para janelas de 14 e 30 dias concluídas.')

import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Carregar previsões e verdadeiros
df = pd.read_csv('results/full_test_set.csv')
# Supondo colunas 'true' e 'pred' no conjunto de teste
true = df['true'].values
pred = df['pred'].values

# Calcular métricas
mse = mean_squared_error(true, pred)
mae = mean_absolute_error(true, pred)
rmse = np.sqrt(mse)

# Exibir resultados
print(f"MSE: {mse:.2f}")
print(f"MAE: {mae:.2f}")
print(f"RMSE: {rmse:.2f}")

# Salvar em CSV
metrics_df = pd.DataFrame({
    'metric': ['MSE','MAE','RMSE'],
    'value': [mse, mae, rmse]
})
metrics_df.to_csv('results/metrics.csv', index=False)
print('Métricas salvas em results/metrics.csv')
```