main

January 22, 2025

1 1. Data Processing

[2]:
```python
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def load_data(file_path):
    """
    Carrega os dados de um arquivo pickle
    """
    with open(file_path, 'rb') as f:
        data = pickle.load(f)
    return data

def flatten_data(data):
    """
    Transforma os dados hierarquicos em um dataframe plano
    """
    flattened_data = []
    for syndrome_id, subjects in data.items():
        for subject_id, images in subjects.items():
            for image_id, embedding in images.items():
                flattened_data.append({
                    "syndrome_id": syndrome_id,
                    "subject_id": subject_id,
                    "image_id": image_id,
                    "embedding": embedding
                })
    return pd.DataFrame(flattened_data)

def validate_embeddings(df):
    """
    Verifica a integridade dos embeddings e remove entradas inconsistentes
    """
    # Verifica se todos os embeddings tem 320 dimensões
    df['embedding_length'] = df['embedding'].apply(lambda x: len(x))
```

```
    inconsistent_embeddings = df[df['embedding_length'] != 320]
    if not inconsistent_embeddings.empty:
        print("Inconsistências detectadas nos embeddings:")
        print(inconsistent_embeddings)
        # Remover embeddings inconsistentes
        df = df[df['embedding_length'] == 320]
    df.drop(columns=['embedding_length'], inplace=True)
    return df

def check_data_integrity(df):
    """
    Exibe informações gerais e verifica dados ausentes
    """
    print("Informações gerais do DataFrame:")
    print(df.info())
    print("\nVerificando valores ausentes:")
    print(df.isnull().sum())
    print("\nExemplo de dados:")
    print(df.head())

def embedding_statistics(df):
    """
    Calcula estatisticas e exibe a distribuicao dos valores nos embeddings
    """
    embeddings_array = np.array(df['embedding'].tolist())
    print("Estatísticas gerais dos embeddings:")
    print("Média:", np.mean(embeddings_array))
    print("Desvio padrão:", np.std(embeddings_array))
    print("Min:", np.min(embeddings_array))
    print("Max:", np.max(embeddings_array))

    # Visualização
    plt.figure(figsize=(10, 6))
    plt.hist(embeddings_array.flatten(), bins=50, alpha=0.7)
    plt.title('Distribuição de Valores nos Embeddings')
    plt.xlabel('Valor')
    plt.ylabel('Frequência')
    plt.savefig('embedding_statistics.png', dpi=300, bbox_inches='tight')
    plt.show()

def exploratory_data_analysis(df):
    """
    Realiza análise exploratória basica dos dados
    """
    print("Número total de imagens:", len(df))
    print("Número total de síndromes únicas:", df['syndrome_id'].nunique())
```

```
    syndrome_counts = df['syndrome_id'].value_counts()
    print("\nDistribuição de imagens por síndrome:")
    print(syndrome_counts)

    # Visualização
    plt.figure(figsize=(10, 6))
    syndrome_counts.plot(kind='bar')
    plt.title('Distribuição de Imagens por Síndrome')
    plt.xlabel('Síndrome ID')
    plt.ylabel('Quantidade de Imagens')
    plt.savefig('exploratory_data_analysis.png', dpi=300, bbox_inches='tight')
    plt.show()

file_path = 'mini_gm_public_v0.1.p'

data = load_data(file_path)
df = flatten_data(data)
df = validate_embeddings(df)
```

[3]:
```
print(df.head())
```

```
  syndrome_id subject_id image_id  \
0   300000082        595     3543
1   300000082       2638     1633
2   300000082     734490   742992
3   300000082       2770     1812
4   300000082       2642     1801

                                           embedding
0  [-0.03718013, 1.741486, 1.2061033, -0.45145318…
1  [2.4249947, 0.17991967, 2.9503245, 0.37993023,…
2  [-1.7893314, -0.21621145, 0.43431538, -0.55232…
3  [-1.1436926, -0.7146209, 0.31948757, 0.4556819…
4  [0.81417066, 1.3274913, 0.84728503, -0.2123311…
```

[3]:
```
check_data_integrity(df)
```

```
Informações gerais do DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1116 entries, 0 to 1115
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   syndrome_id  1116 non-null   object
 1   subject_id   1116 non-null   object
 2   image_id     1116 non-null   object
 3   embedding    1116 non-null   object
```

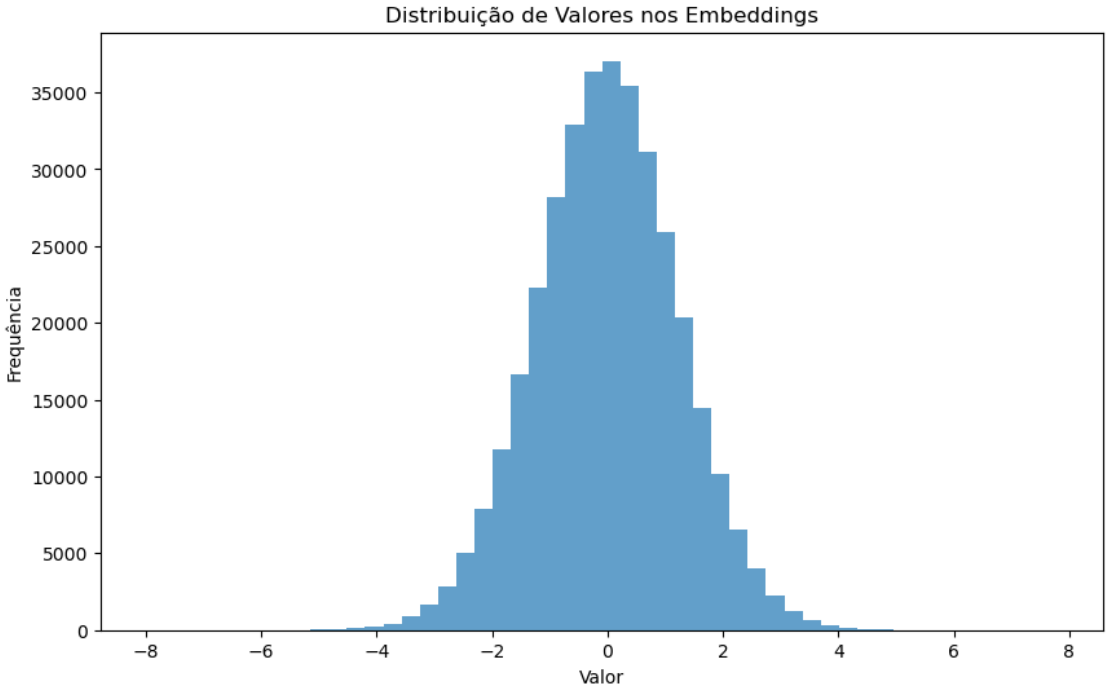
```
dtypes: object(4)
memory usage: 35.0+ KB
None

Verificando valores ausentes:
syndrome_id    0
subject_id     0
image_id       0
embedding      0
dtype: int64

Exemplo de dados:
  syndrome_id subject_id image_id  \
0   300000082        595     3543
1   300000082       2638     1633
2   300000082     734490   742992
3   300000082       2770     1812
4   300000082       2642     1801

                                           embedding
0  [-0.03718013, 1.741486, 1.2061033, -0.45145318…
1  [2.4249947, 0.17991967, 2.9503245, 0.37993023,…
2  [-1.7893314, -0.21621145, 0.43431538, -0.55232…
3  [-1.1436926, -0.7146209, 0.31948757, 0.4556819…
4  [0.81417066, 1.3274913, 0.84728503, -0.2123311…
```

A execução da célula acima confirma que o DataFrame possui 1116 registros e 4 colunas (syndrome_id, subject_id, image_id e embedding), sem valores ausentes em nenhuma delas. A coluna embedding contém listas de valores numéricos tratados como objetos, e todas as colunas estão consistentes com os tipos esperados.

A amostra apresentada mostra identificadores numéricos para as colunas de síndromes, sujeitos e imagens, enquanto a coluna embedding contém vetores bem estruturados.

[4]:
```
embedding_statistics(df)
```

```
Estatísticas gerais dos embeddings:
Média: -0.00010721333
Desvio padrão: 1.2335585
Min: -7.9867287
Max: 7.7985287
```



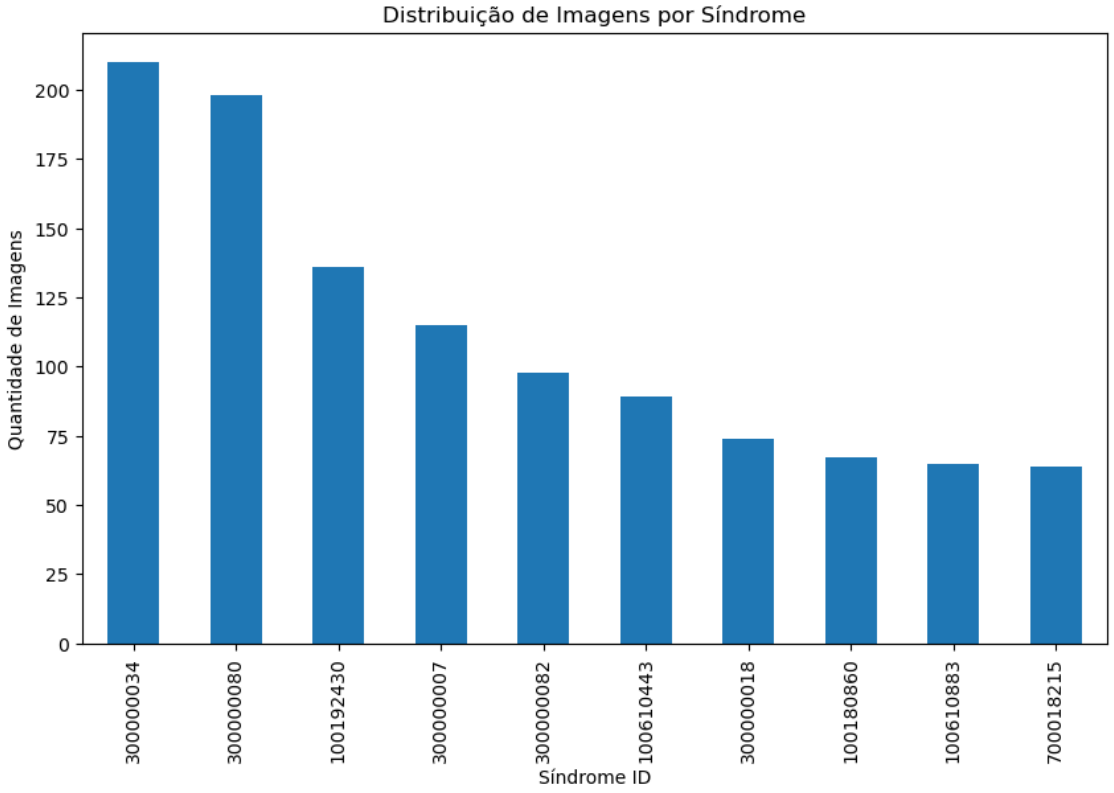
A execução da função acima revelou que os embeddings possuem uma média de -0.00010721333, indicando que os dados estão bem centralizados em torno de zero. O desvio padrão é 1.2335585, demonstrando uma dispersão consistente e apropriada para análises de aprendizado de máquina. Os valores mínimos e máximos são -7.9867287 e 7.7985287, respectivamente, sugerindo uma amplitude equilibrada sem indícios de valores extremos que possam comprometer o modelo.

O histograma gerado mostra que a distribuição dos valores dos embeddings segue um formato aproximadamente normal, com maior concentração em torno de zero e uma dispersão visível, o que pode ser útil para a captura de padrões pelo modelo. Essa boa distribuição dos embeddings, centrada e com variabilidade adequada, indica que os dados estão em condições favoráveis para serem utilizados nas próximas etapas.

[5]:
```
exploratory_data_analysis(df)
```

```
Número total de imagens: 1116
Número total de síndromes únicas: 10

Distribuição de imagens por síndrome:
syndrome_id
300000034    210
300000080    198
100192430    136
300000007    115
300000082     98
100610443     89
300000018     74
```

```
100180860      67
100610883      65
700018215      64
Name: count, dtype: int64
```



A execução da função acima revelou que o conjunto de dados contém um total de 1116 imagens distribuídas entre 10 síndromes únicas. A análise da distribuição de imagens por síndrome mostra que há um desequilíbrio significativo no número de imagens por categoria. A síndrome 30000034 possui a maior quantidade de imagens, com 210 registros, seguida pela síndrome 30000080, com 198 imagens. Outras síndromes, como 700018215 e 100610883, possuem apenas 64 e 65 imagens, respectivamente, sendo as menores categorias no conjunto.

O gráfico de barras reflete claramente esse desequilíbrio, destacando que algumas síndromes dominam o conjunto de dados, enquanto outras têm representações limitadas. Este desequilíbrio pode impactar o desempenho do modelo de classificação, especialmente em categorias com menor representação, potencialmente levando a problemas de generalização.

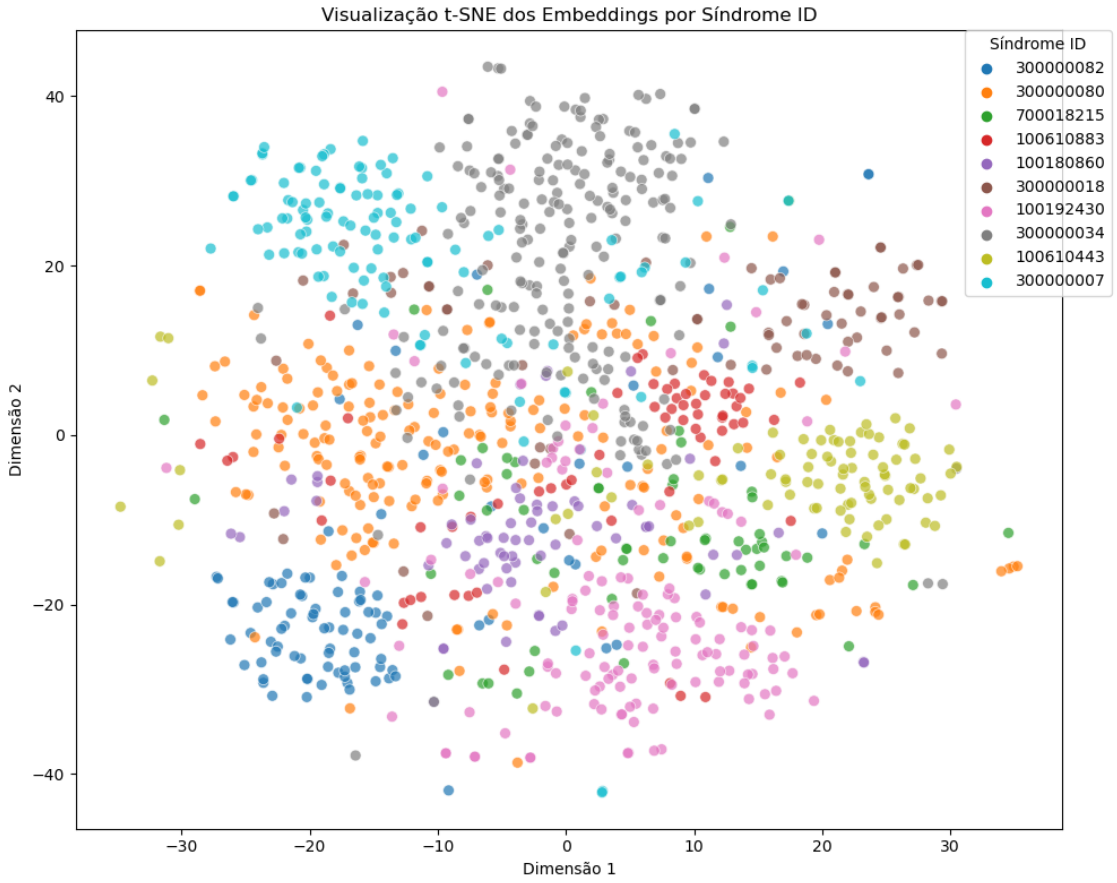
Com base nesses resultados, é importante considerar técnicas para mitigar os efeitos desse desbalanceamento, como reamostragem (oversampling ou undersampling) ou o uso de métodos que ponderem as classes de forma adequada durante o treinamento do modelo.

2 2. Data Visualization

[6]:
```
from sklearn.manifold import TSNE
import seaborn as sns

def visualize_embeddings_tsne(df, perplexity=30, learning_rate=200,␣
↪random_state=42):
    """
    Reduz a dimensionalidade dos embeddings para 2D usando t-SNE e visualiza os␣
↪resultados
    """
    # Extrai os embeddings e os rotulos
    embeddings = np.array(df['embedding'].tolist())
    syndrome_ids = df['syndrome_id']

    # Aplica tSNE
    tsne = TSNE(n_components=2, perplexity=perplexity,␣
↪learning_rate=learning_rate, random_state=random_state)
    embeddings_2d = tsne.fit_transform(embeddings)

    # Cria um dataframe para os resultados
    tsne_df = pd.DataFrame({
        'Dimension 1': embeddings_2d[:, 0],
        'Dimension 2': embeddings_2d[:, 1],
        'Syndrome ID': syndrome_ids
    })

    # Visualiza
    plt.figure(figsize=(10, 8))
    sns.scatterplot(
        data=tsne_df,
        x='Dimension 1',
        y='Dimension 2',
        hue='Syndrome ID',
        palette='tab10',
        s=50,
        alpha=0.7
    )
    plt.title('Visualização t-SNE dos Embeddings por Síndrome ID')
    plt.xlabel('Dimensão 1')
    plt.ylabel('Dimensão 2')
    plt.legend(loc='best', title='Síndrome ID', bbox_to_anchor=(1.05, 1),␣
↪borderaxespad=0.)
    plt.tight_layout()
    plt.savefig('visualize_embeddings_tsne.png', dpi=300, bbox_inches='tight')
    plt.show()
```

```
visualize_embeddings_tsne(df)
```

```
D:\Downloads2\Anaconda\download\lib\site-
packages\sklearn\manifold\_t_sne.py:780: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
```



A análise da visualização gerada pelo t-SNE revela importantes insights sobre os embeddings e sua relação com as síndromes. Algumas síndromes, como 100610443 (amarelo esverdeado) e 30000082 (azul), apresentam clusters relativamente bem definidos, indicando que seus embeddings possuem separabilidade no espaço de 2D, o que pode facilitar a tarefa de classificação. Por outro lado, outras síndromes, como 700018215 (verde) e 100180860 (roxo), mostram maior dispersão e sobreposição com outras classes, o que pode dificultar o aprendizado do modelo.

A sobreposição entre as classes sugere que os embeddings de algumas síndromes não são suficientemente discriminados, o que pode levar a confusões durante a classificação. Isso é particularmente evidente em síndromes como 700018215, que compartilham áreas densas no gráfico com outras classes adjacentes.

Esses padrões têm implicações diretas para a tarefa de classificação. Síndromes que formam clusters bem definidos podem ser classificadas com maior precisão, enquanto aquelas com maior dispersão

ou sobreposição provavelmente impactarão negativamente métricas como a precisão e o F1-Score. Para lidar com esses desafios, pode ser necessário explorar modelos mais sofisticados, como SVMs com kernels não lineares ou redes neurais profundas, que têm maior capacidade de capturar relações complexas entre os dados.

3 3. Classification Task

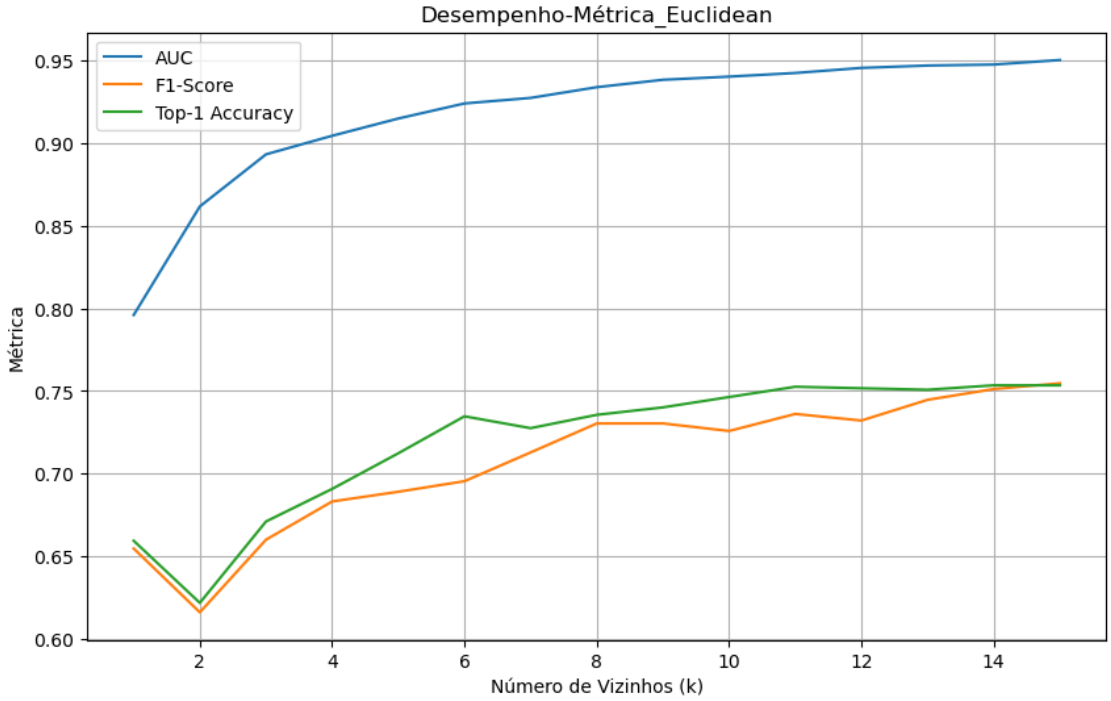
[7]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_auc_score, f1_score, accuracy_score,␣
 ↪top_k_accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def evaluate_model(y_true, y_pred, y_prob, k):
    """
    Avalia o desempenho do modelo com base em diferentes metricas
    """
    auc = roc_auc_score(y_true, y_prob, multi_class='ovr')
    f1 = f1_score(y_true, y_pred, average='weighted')
    top_k_acc = top_k_accuracy_score(y_true, y_prob, k=k)
    return auc, f1, top_k_acc

def knn_classification(df, k_values=range(1, 16), metric='euclidean'):
    """
    Realiza classificacao KNN com validação cruzada e retorna métricas de␣
 ↪desempenho
    """
    embeddings = np.array(df['embedding'].tolist())
    labels = df['syndrome_id']
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    results = {k: {"AUC": [], "F1-Score": [], "Top-1 Accuracy": []} for k in␣
 ↪k_values}

    for train_index, test_index in skf.split(embeddings, labels):
        X_train, X_test = embeddings[train_index], embeddings[test_index]
        y_train, y_test = labels.iloc[train_index], labels.iloc[test_index]

        for k in k_values:
            # Define KNN
            knn = KNeighborsClassifier(n_neighbors=k, metric=metric)
            knn.fit(X_train, y_train)

            # Predicoes
            y_pred = knn.predict(X_test)
```

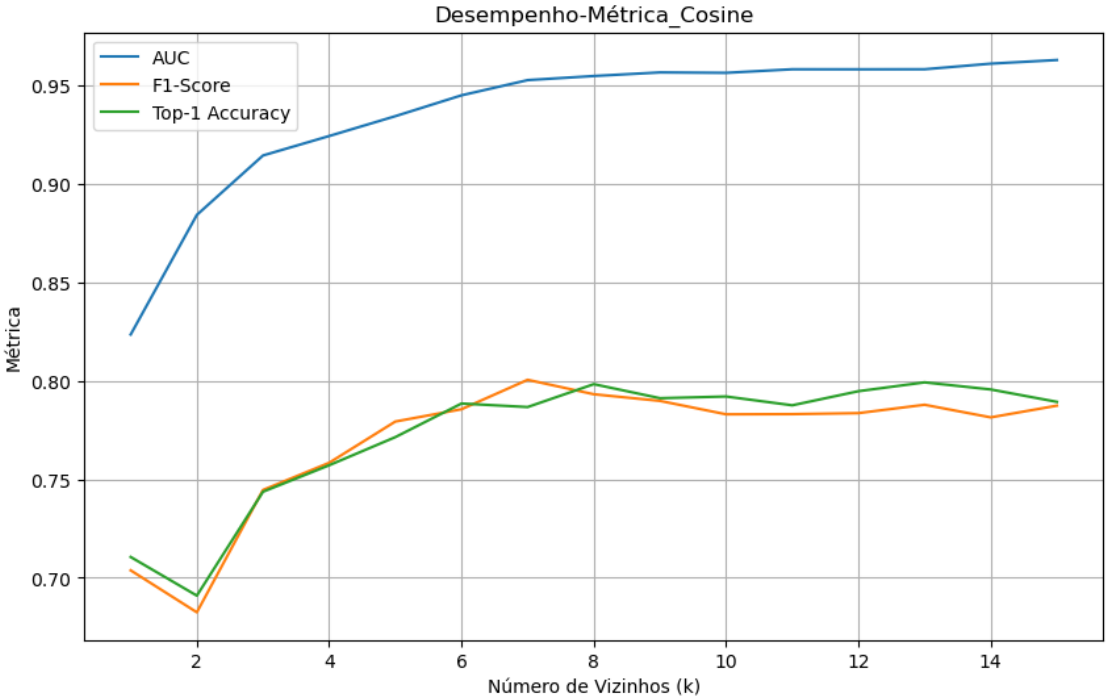
```
            y_prob = knn.predict_proba(X_test)

            # Avaliacao do modelo
            auc, f1, top_k_acc = evaluate_model(y_test, y_pred, y_prob, k=1)
            results[k]["AUC"].append(auc)
            results[k]["F1-Score"].append(f1)
            results[k]["Top-1 Accuracy"].append(top_k_acc)

    # Agrega resultados
    aggregated_results = {k: {metric: np.mean(values[metric]) for metric in␣
 ↪values} for k, values in results.items()}
    return aggregated_results

def compare_metrics(df, k_values=range(1, 16)):
    """
    Compara as metricas de classificacao para as distâncias Euclidean e Cosine
    """
    print("Calculando para Métrica Euclidean...")
    results_euclidean = knn_classification(df, k_values=k_values,␣
 ↪metric='euclidean')

    print("Calculando para Métrica Cosine...")
    results_cosine = knn_classification(df, k_values=k_values, metric='cosine')

    return results_euclidean, results_cosine

k_values = range(1, 16)
results_euclidean, results_cosine = compare_metrics(df, k_values=k_values)

euclidean_df = pd.DataFrame(results_euclidean).T
cosine_df = pd.DataFrame(results_cosine).T

print("\nResultados - Métrica Euclidean")
print(euclidean_df)

print("\nResultados - Métrica Cosine")
print(cosine_df)

# Visualizacao
def plot_results(results, title):
    """
    Plota os resultados das metricas para diferentes valores de k
    """
    ks = list(results.keys())
    auc = [results[k]["AUC"] for k in ks]
    f1 = [results[k]["F1-Score"] for k in ks]
    top1 = [results[k]["Top-1 Accuracy"] for k in ks]
```

```
    plt.figure(figsize=(10, 6))
    plt.plot(ks, auc, label="AUC")
    plt.plot(ks, f1, label="F1-Score")
    plt.plot(ks, top1, label="Top-1 Accuracy")
    plt.title(title)
    plt.xlabel("Número de Vizinhos (k)")
    plt.ylabel("Métrica")
    plt.legend()
    plt.grid()
    plt.savefig('plot_results_' + title + '.png', dpi=300, bbox_inches='tight')
    plt.show()

plot_results(results_euclidean, "Desempenho-Métrica_Euclidean")

plot_results(results_cosine, "Desempenho-Métrica_Cosine")
```

```
Calculando para Métrica Euclidean…
Calculando para Métrica Cosine…

Resultados - Métrica Euclidean
         AUC  F1-Score  Top-1 Accuracy
1   0.796021  0.654707        0.659411
2   0.861734  0.616099        0.621823
3   0.893266  0.660055        0.671067
4   0.904500  0.683189        0.690798
5   0.914949  0.689078        0.712355
6   0.924069  0.695446        0.734733
7   0.927422  0.712866        0.727550
8   0.933907  0.730387        0.735626
9   0.938392  0.730408        0.740130
10  0.940303  0.725837        0.746396
11  0.942501  0.736153        0.752606
12  0.945571  0.732170        0.751705
13  0.947016  0.744711        0.750804
14  0.947594  0.751219        0.753483
15  0.950351  0.754720        0.753467

Resultados - Métrica Cosine
         AUC  F1-Score  Top-1 Accuracy
1   0.823559  0.703736        0.710521
2   0.884325  0.682414        0.690886
3   0.914518  0.744595        0.743678
4   0.924401  0.758387        0.757127
5   0.934484  0.779342        0.771421
6   0.945076  0.785593        0.788489
7   0.952809  0.800547        0.786704
```

8	0.954891	0.793211	0.798335
9	0.956719	0.789834	0.791208
10	0.956507	0.783013	0.792061
11	0.958314	0.783105	0.787588
12	0.958297	0.783612	0.794755
13	0.958324	0.787870	0.799244
14	0.961168	0.781454	0.795640
15	0.963019	0.787409	0.789342





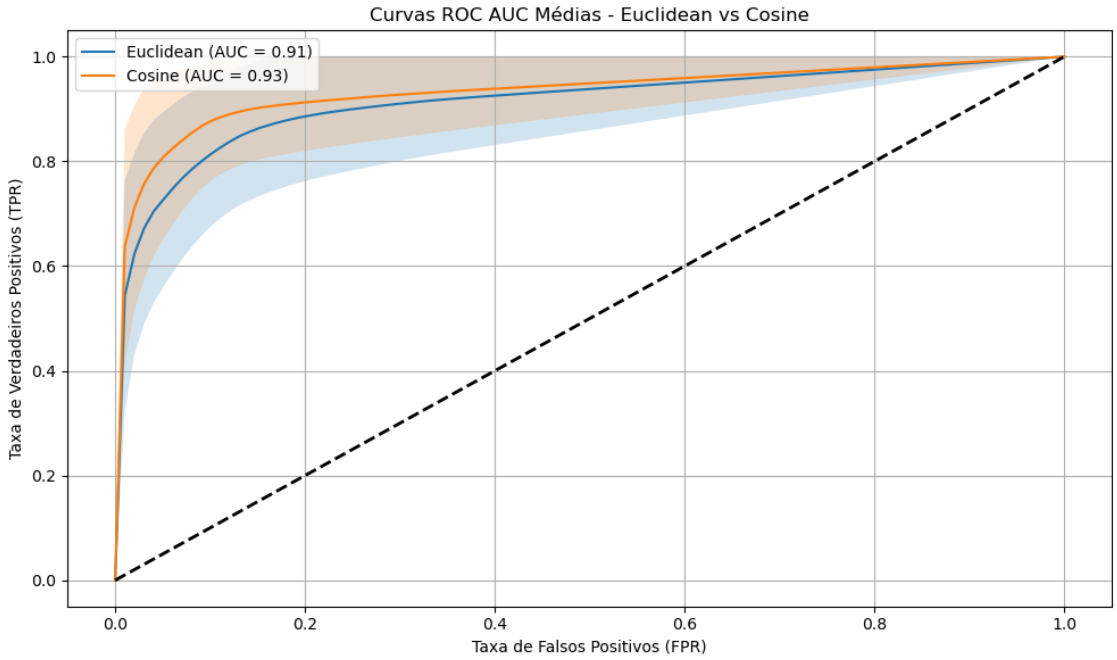
A análise dos resultados mostra que a métrica Cosine apresenta desempenho superior. Para ambos casos, o desempenho continua a melhorar com o aumento de k, indicando maior eficácia na captura de relações globais entre os embeddings. As diferenças de desempenho são explicadas pela natureza das métricas: a Euclidean mede a distância linear, sendo mais sensível à separação global dos dados, enquanto a Cosine mede similaridade angular, sendo mais robusta para diferenças de magnitude nos vetores. Para esse problema específico deste projeto, a métrica Cosine é a mais adequada a partir dos resultados obtidos.

4 4. Metrics and Evaluation

[8]:
```
from sklearn.metrics import roc_curve, auc

def plot_roc_curves(df, k, metric, ax):
    """
    Calcula e plota a curva ROC AUC media para a metrica especificada
    """
    embeddings = np.array(df['embedding'].tolist())
    labels = pd.get_dummies(df['syndrome_id']).values
    skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    mean_fpr = np.linspace(0, 1, 100)
    tprs = []
    aucs = []
```

```
    for train_index, test_index in skf.split(embeddings, df['syndrome_id']):
        X_train, X_test = embeddings[train_index], embeddings[test_index]
        y_train, y_test = labels[train_index], labels[test_index]

        knn = KNeighborsClassifier(n_neighbors=k, metric=metric)
        knn.fit(X_train, np.argmax(y_train, axis=1))
        y_prob = knn.predict_proba(X_test)

        # Curvas ROC para cada classe
        for i in range(y_test.shape[1]):
            fpr, tpr, _ = roc_curve(y_test[:, i], y_prob[:, i])
            interp_tpr = np.interp(mean_fpr, fpr, tpr)
            interp_tpr[0] = 0.0
            tprs.append(interp_tpr)
            roc_auc = auc(fpr, tpr)
            aucs.append(roc_auc)

    # Media e intervalo da curva ROC
    mean_tpr = np.mean(tprs, axis=0)
    mean_tpr[-1] = 1.0
    mean_auc = auc(mean_fpr, mean_tpr)

    # Plotagem
    ax.plot(mean_fpr, mean_tpr, label=f'{metric.capitalize()} (AUC = {mean_auc:.
↪2f})')
    ax.fill_between(
        mean_fpr,
        np.maximum(mean_tpr - np.std(tprs, axis=0), 0),
        np.minimum(mean_tpr + np.std(tprs, axis=0), 1),
        alpha=0.2,
    )
    return mean_auc

# Plotando as curvas para Cosine e Euclidean
fig, ax = plt.subplots(figsize=(10, 6))
auc_euclidean = plot_roc_curves(df, k=5, metric='euclidean', ax=ax)
auc_cosine = plot_roc_curves(df, k=5, metric='cosine', ax=ax)

# Configuracao do gráfico
ax.plot([0, 1], [0, 1], 'k--', lw=2)
ax.set_title('Curvas ROC AUC Médias - Euclidean vs Cosine')
ax.set_xlabel('Taxa de Falsos Positivos (FPR)')
ax.set_ylabel('Taxa de Verdadeiros Positivos (TPR)')
ax.legend()
plt.grid()
plt.tight_layout()
plt.savefig('plot_roc_curves.png', dpi=300, bbox_inches='tight')
```

```
plt.show()

print("\nTabela de Resultados - Métrica Euclidean")
print(euclidean_df)

print("\nTabela de Resultados - Métrica Cosine")
print(cosine_df)
```




```
Tabela de Resultados - Métrica Euclidean
         AUC  F1-Score  Top-1 Accuracy
1   0.796021  0.654707        0.659411
2   0.861734  0.616099        0.621823
3   0.893266  0.660055        0.671067
4   0.904500  0.683189        0.690798
5   0.914949  0.689078        0.712355
6   0.924069  0.695446        0.734733
7   0.927422  0.712866        0.727550
8   0.933907  0.730387        0.735626
9   0.938392  0.730408        0.740130
10  0.940303  0.725837        0.746396
11  0.942501  0.736153        0.752606
12  0.945571  0.732170        0.751705
13  0.947016  0.744711        0.750804
14  0.947594  0.751219        0.753483
15  0.950351  0.754720        0.753467
```

```
Tabela de Resultados - Métrica Cosine
         AUC  F1-Score  Top-1 Accuracy
1   0.823559  0.703736        0.710521
2   0.884325  0.682414        0.690886
3   0.914518  0.744595        0.743678
4   0.924401  0.758387        0.757127
5   0.934484  0.779342        0.771421
6   0.945076  0.785593        0.788489
7   0.952809  0.800547        0.786704
8   0.954891  0.793211        0.798335
9   0.956719  0.789834        0.791208
10  0.956507  0.783013        0.792061
11  0.958314  0.783105        0.787588
12  0.958297  0.783612        0.794755
13  0.958324  0.787870        0.799244
14  0.961168  0.781454        0.795640
15  0.963019  0.787409        0.789342
```

As curvas ROC AUC do gráfico acima mostram que ambas as métricas apresentam desempenhos excelentes para a classificação de síndromes. A métrica Cosine alcançou uma AUC média de 0.93, enquanto a métrica Euclidean apresentou uma AUC de 0.91. A curva da métrica Cosine está ligeiramente mais próxima do canto superior esquerdo, indicando melhor separação entre as classes. Além disso, a região sombreada, que representa a variação entre os folds de validação cruzada, aparenta ser menor para a métrica Cosine, sugerindo maior consistência nos resultados, enquanto a métrica Euclidean apresenta maior variabilidade em alguns pontos. Essa diferença pode ser explicada pela robustez da métrica Cosine em capturar similaridades angulares entre os vetores, especialmente útil quando os embeddings apresentam variações em suas magnitudes.

4.1 Conclusão

Os resultados demonstram que tanto a métrica Cosine quanto a Euclidean são altamente eficazes na tarefa de classificação. As curvas ROC AUC médias reforçam o alto desempenho de ambas as métricas, com uma ligeira vantagem para a Cosine em termos de separação entre as classes. Além disso, dentre o intervalo de valores possíveis para k, quanto maior o k melhor o desempenho. Por fim, é possível se alcançar bons resultados para a tarefa de classificação com as configurações propostas para esse experimentos.

[ ]: