



**CEPEDI - RESTIC 36**

## **Relatório Técnico: Implementação e Análise do Algoritmo de K-means**

Membros da equipe: Fabrício Soares Oliveira, Geisa Almeida dos Santos

Eunápolis 01 de dezembro de 2024

## RESUMO

O projeto tem como foco a análise de dados utilizando o algoritmo K-means, com o objetivo de implementá-lo no dataset "Human Activity Recognition Using Smartphones". A proposta é explorar os dados, aplicar o K-means e identificar padrões ou agrupamentos de atividades.

A base de dados utilizada contém informações coletadas por sensores de acelerômetro e giroscópio de smartphones de 30 voluntários durante a realização de atividades cotidianas. O projeto abrange desde a análise exploratória dos dados até a definição do número ideal de clusters e a visualização dos resultados obtidos.

## INTRODUÇÃO

Este relatório apresenta uma breve análise sobre o algoritmo K-means, um método de aprendizado de máquina não supervisionado amplamente utilizado em tarefas de agrupamento de dados. Ele é de grande importância em áreas como marketing, detecção de padrões em dados sensoriais, e agrupamento de documentos ou imagens com características semelhantes, onde é frequentemente aplicado.

O projeto tem como objetivo desenvolver um algoritmo capaz de identificar padrões ou agrupamentos de atividades. Para isso, será necessário definir a quantidade de agrupamentos ( $K$ ), os centróides e outros métodos fundamentais para a execução eficiente do programa.

Espera-se que o algoritmo produza os  $K$  clusters finais, com os dados agrupados e seus respectivos centróides, indicando que os centróides não estão mais mudando significativamente entre as iterações ou que o número máximo de iterações foi atingido. Apesar de ser eficiente para grandes volumes de dados, o algoritmo apresenta algumas limitações, como a necessidade de definir previamente o número de clusters ( $K$ ) e sua sensibilidade a outliers.

Entretanto, essas limitações não comprometem sua viabilidade, pois o K-means continua sendo uma ferramenta poderosa para revelar padrões ocultos em

conjuntos de dados complexos. Para o desenvolvimento deste programa, utilizaremos o Colab, algumas bibliotecas e a base de dados mencionada anteriormente. Por fim, será realizada a análise e o tratamento dos dados.

## **Metodologia**

Para iniciar a elaboração do K-means, um algoritmo de agrupamento, é necessário realizar algumas etapas preliminares, como a importação da base de dados que será utilizada. Primeiramente, cria-se uma pasta no Google Drive para armazenar os arquivos, ressaltando que, neste projeto, trabalharemos com mais de um arquivo.

Em seguida, abrimos o Colab e criamos um novo notebook. Para acessar a base de dados, conectamos o Colab ao Drive, permitindo o acesso aos arquivos salvos anteriormente. Depois disso, importa-se as bibliotecas essenciais, como Pandas e Numpy.

O Pandas é fundamental para a manipulação de dados tabulares, sendo amplamente utilizado para leitura e manipulação de formatos como CSV, SQL e JSON. Já o Numpy é ideal para computação numérica, com foco em operações eficientes com arrays e matrizes, além de oferecer funções matemáticas de alto desempenho, como somas e cálculos vetorizados.

Com a conexão ao Drive e a importação das bibliotecas concluídas, os arquivos podem ser localizados e manipulados com eficiência. O comando `gdown` é especialmente útil para carregar arquivos grandes ou datasets diretamente no Colab, eliminando a necessidade de uploads manuais. Além disso, o comando `from google.colab import drive; drive.mount('/content/drive')` é essencial, pois permite o acesso à conta do Google Drive, facilitando a manipulação dos arquivos armazenados.

No entanto, desta vez, precisamos realizar o processo de carregamento mais vezes, devido ao uso de arquivos distintos. Há a necessidade de fazer a leitura tanto de `X_treino` quanto de `Y_treino`, além do carregamento de outro arquivo, o `features_names_path`, que contém informações sobre os nomes das variáveis.

Ao carregar as bases de dados, identificamos a necessidade de atribuir nomes às colunas. Esse processo facilita a compreensão dos dados e evita

possíveis erros, como falhas ao acessar uma coluna específica. O procedimento realizado será ilustrado na imagem a seguir.

Figura 1 - Inserção de nomes

```
#Inserção dos nomes das variáveis

features_names_path = "/content/drive/MyDrive/COLAB ARQUIVOS/UCI HAR Dataset/features.txt"
features_names = pd.read_csv(features_names_path, delim_whitespace=True, header=None)[1].tolist()

# Aplicação dos nomes
x_treino.columns = features_names
print(x_treino.head())
```

Fonte: Elaborada pelos autores (2024)

Com essas etapas concluídas, passamos à verificação de valores nulos ou ausentes, processo realizado em todos os dados de X\_treino e Y\_treino ste. Para garantir a eficiência do algoritmo, removemos os valores ausentes ou nulos, pois sua presença pode comprometer a precisão dos cálculos de distância e a qualidade dos clusters gerados. A seguir, uma imagem ilustrativa do processo.

Figura 1 - Valores nulos

```
# Verificando valores nulos
x_treino.isnull().sum()
y_treino.isnull().sum()

# Remover linhas com valores ausentes (NaN)
x_treino.dropna(inplace=True)
y_treino.dropna(inplace=True)
```

Fonte: Elaborada pelos autores (2024)

Logo depois, iniciamos a visualização dos dados, começando pela análise da distribuição das atividades presentes no conjunto de dados y\_treino. Em seguida, realizamos o mesmo processo com os demais dados. O objetivo é identificar possíveis desequilíbrios nas classes e verificar se há uma quantidade significativa de amostras para cada atividade.

Já que desequilíbrios podem impactar a performance do K-means, se uma atividade (classe) ocorre significativamente mais vezes que as outras, o algoritmo pode priorizar clusters para essa classe majoritária, ignorando aquelas

com menos amostras. Isso pode levar a um modelo menos eficiente. Segue imagem ilustrativa.

Figura 1 - Distribuição

```
#Distribuição do dados de Y_treino
plt.figure(figsize=(10, 6))

sns.countplot(x='Activity', data=y_treino)
plt.title('Distribuição das Atividades')
plt.xlabel('Atividade')
plt.ylabel('Frequência')
plt.xticks(rotation=45)
plt.show()
print(y_treino['Activity'].value_counts())
```

Fonte: Elaborada pelos autores (2024)

No caso dos dados ilustrados acima, não foram identificados desequilíbrios significativos, por isso não houve necessidade de tratamento. Contudo, caso houvesse desequilíbrio, uma opção seria realizar uma reamostragem, que visa ajustar a distribuição das classes no conjunto de dados. Uma das técnicas mais populares para isso é o SMOTE (Synthetic Minority Over-sampling Technique), que gera exemplos sintéticos para a classe minoritária, em vez de apenas duplicar os exemplos existentes.

Diferentemente do trabalho com Regressão Linear, no K-means não foi necessário identificar a correlação entre as variáveis, pois já estávamos lidando com o banco de dados dividido entre X\_treino e Y\_treino. Esse fator nos dispensou da necessidade de realizar a correlação entre variáveis. No entanto, isso trouxe um desafio adicional, ter que trabalhar com bases de dados individualmente.

Um método de extrema importância para o algoritmo K-means é o tratamento de outliers. Por isso, foi essencial verificar a existência desses pontos fora do padrão em nossos dados, já que eles podem distorcer os resultados e prejudicar a eficácia da clusterização. Por essa razão, realizamos a verificação da presença de outliers em X\_treino, como ilustrado na imagem a seguir.

Figura 4 - Outliers

```
# Verificando a existe de outliers em X-treino

amostra_x_treino = x_treino.columns[:36]

plt.figure(figsize=(10, 5))
x_treino[amostra_x_treino].boxplot()
plt.title('Amostra dos Dados de Treinamento')
plt.xticks(rotation=45)
plt.show()
```

Fonte: Elaborada pelos autores (2024)

Através dessas verificações, conseguimos identificar a presença de outliers em ambos os conjuntos de dados, com alguns pontos mais agrupados e outros mais dispersos. Como estávamos lidando com vários dados, optamos por realizar o tratamento nas 36 colunas, mas também seria possível direcionar o tratamento para colunas específicas. Contudo, essa abordagem poderia tornar o código mais complexo, visto que seriam muitas as colunas a serem tratadas de forma individual.

O tratamento de outliers foi essencial, pois não queríamos que esses pontos impactassem negativamente os resultados finais do nosso modelo. Entre as alternativas disponíveis, escolhemos remover os outliers de todas as colunas de X\_treino utilizando o intervalo interquartil (IQR). Esse método foi escolhido por ser amplamente utilizado quando os dados se distribuem de maneira aproximadamente normal.

Como o próprio nome sugere, o IQR divide os dados em quatro partes, sendo que cada uma delas contém 25% dos dados. Ele se baseia nos quartis e não na média, tornando-se mais eficaz quando os dados são assimétricos. Com isso, ele consegue identificar valores atípicos, ou seja, os outliers, como ilustrado na figura abaixo.

Figura 1 - Tratamento

```
#Realizadno o trataemnto dos outliers de Xtreno.

Q3 = x_treino.quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
x_treino_outliers = x_treino[~((x_treino < (Q1 - 1.5 * IQR)) | (x_treino > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Fonte: Elaborada pelos autores (2024)

Logo em seguida, buscamos realizar a normalização dos dados, processo que foi executado em `X_treino`. Esse procedimento é de extrema importância para o algoritmo K-means, pois ele é sensível às escalas das variáveis. Se as variáveis tiverem escalas muito diferentes, algumas podem dominar o cálculo das distâncias, prejudicando o desempenho do algoritmo.

Durante a normalização, avaliamos três métodos para identificar a melhor opção para nosso caso. Os métodos testados foram: `StandardScaler()`, `RobustScaler()` e `MaxAbsScaler()`. Com o primeiro método, obtemos um resultado razoável, com o segundo tivemos um ótimo desempenho, mas ainda assim testamos um terceiro método. Contudo, o `MaxAbsScaler()` não produziu resultados tão significativos em comparação aos anteriores.

Dentre essas opções, o método que melhor atendeu ao nosso problema foi o `RobustScaler()`, pois com ele conseguimos uma dimensionalidade de 0.9594, bastante próxima de 1. Além disso, percebemos que a utilização desse método resultou em 8 clusters, o que é um ótimo resultado, já que nossa base possui 6 rótulos. A seguir, a imagem ilustrando o método utilizado.

Figura 1 - Normalização

```
scaler = RobustScaler()
features_scaled = scaler.fit_transform(x_treino)
x_treino_normalized2 = pd.DataFrame(features_scaled)
print(x_treino_normalized2.head())
```

Fonte: Elaborada pelos autores (2024)

Acreditamos que o método `StandardScaler()` não obteve o melhor resultado devido à sua sensibilidade a outliers. Diferente dele, o `RobustScaler()` se mostrou bem resistente a outliers, porém não garante uma média de 0 e desvio padrão de 1, como no caso do `StandardScaler`. Já o `MaxAbsScaler()` escala os dados entre -1 e 1, dividindo-os pelo valor absoluto máximo. Esse método mantém a dispersão dos dados e é mais adequado para dados esparsos.

Após concluir o processo de normalização e tratamento de outliers em todos os dados, iniciamos a aplicação do PCA (Análise de Componentes Principais).

O principal objetivo do PCA é simplificar a estrutura dos dados, preservando a maior parte de sua variabilidade, reduzindo a redundância e, conseqüentemente, o custo computacional ao trabalhar com menos dimensões. A seguir, está a imagem do código utilizado.

Figura 1 - Dimensionalidade

```
# Redução de dimensionalidade para visualização (2D)
pca = PCA(n_components= 2)
x_pca = pca.fit_transform(x_treino_normatized2)
```

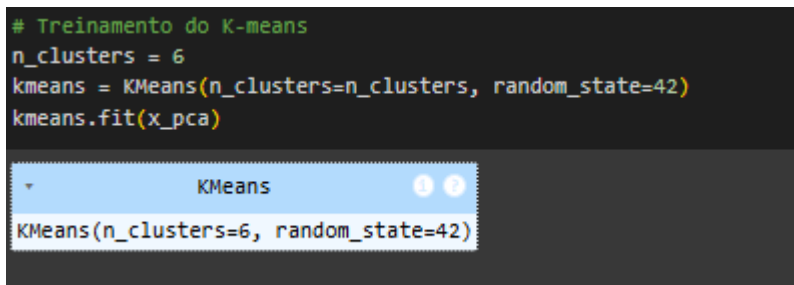
Fonte: Elaborada pelos autores (2024)

Utilizamos o método com dois componentes, pois os dados podem ser projetados em um gráfico 2D, o que é útil para identificar padrões, clusters ou outliers. Os métodos de normalização mencionados anteriormente são extremamente importantes, pois garantem que todas as variáveis tenham o mesmo peso, já que o PCA é sensível à escala das variáveis.

Em seguida, iniciamos o treinamento do nosso K-means, no qual passamos como parâmetro o valor 6, correspondente aos outliers, e utilizamos nosso conjunto de dados Y, que contém 6 rótulos, para agrupar os dados em 6 clusters. Essa escolha também foi baseada na análise do resultado da nossa Silhouette Score. A seguir, a imagem do treinamento.

Figura 1 - Treinamento

```
# Treinamento do K-means
n_clusters = 6
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
kmeans.fit(x_pca)
```



Fonte: Elaborada pelos autores (2024)



Após realizar o treinamento, buscamos visualizar como os pontos foram agrupados no espaço reduzido pelo PCA. A separação visual reflete a eficácia do algoritmo em distinguir os grupos. Cada ponto pertence a um dos 6 clusters definidos anteriormente, sendo possível identificar as classes por cores. A seguir, o código referente à plotagem.

Figura 1 - Visualização

```
# Visualização dos clusters
labels = kmeans.labels_
plt.figure(figsize=(10, 6))
plt.scatter(x_pca[:, 0], x_pca[:, 1], c=labels, cmap='viridis', s=10)
plt.title('Visualização dos Clusters (K-Means++)')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.colorbar(label='Cluster')
plt.show()
```

Fonte: Elaborada pelos autores (2024)

Logo em seguida, utilizamos o método Silhouette, que avalia a qualidade dos clusters em nosso modelo, medindo o quão bem cada ponto está associado ao seu cluster em comparação com os outros clusters. Ele realiza cálculos para diferentes valores de K, e através dele conseguimos identificar o melhor K, ou seja, o número ideal de clusters. A seguir, o código.

Figura 1 - Verificação de K

```
silhouette_scores = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    clusters = kmeans.fit_predict(x_pca_roubust)
    score = silhouette_score(x_pca_roubust, clusters)
    silhouette_scores.append(score)
    plt.figure(figsize=(8, 6))
plt.plot(K_range, silhouette_scores, '-o')
plt.title("Silhouette Score para diferentes valores de K")
plt.xlabel("Número de Clusters (K)")
plt.ylabel("Silhouette Score")
plt.show()
best_k = K_range[np.argmax(silhouette_scores)]
print(f"\nNúmero ideal de clusters com base no Silhouette Score: K={best_k}")
print (silhouette_scores)
```

Fonte: Elaborada pelos autores (2024)

Após obter o resultado da Silhouette Score, utilizamos o método descrito para a aplicação do K-Means com cálculo da Silhouette Score, pois ele é importante para realizar um agrupamento eficaz e avaliar a qualidade dos clusters gerados. Ele nos permite encontrar o número ideal de clusters e melhorar a qualidade da segmentação dos dados. A seguir, a imagem do código.

Figura 1 - Redução de dimensionalidade

```
optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
clusters = kmeans.fit_predict(x_pca_roubust)
silhouette_avg = silhouette_score(x_pca_roubust, clusters)
print(f"Silhouette Score após redução de dimensionalidade: {silhouette_avg:.4f}")

Silhouette Score após redução de dimensionalidade: 0.9594
```

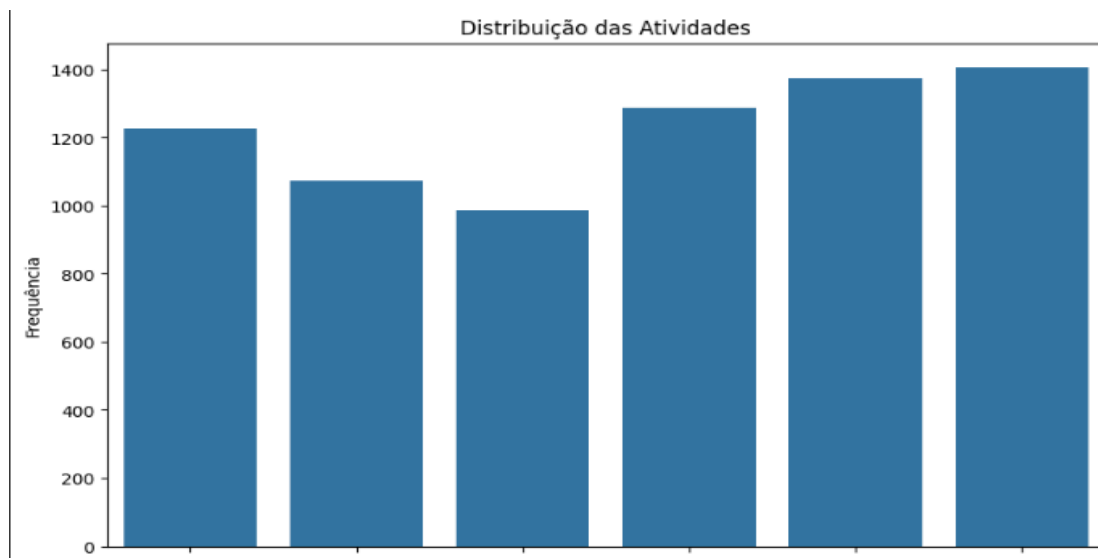
Fonte: Elaborada pelos autores (2024)

## RESULTADOS

O objetivo do projeto tinha como finalidade realizar uma análise exploratória dos dados para determinar o número ideal de clusters e visualizar os resultados, utilizando o algoritmo K-Means com o auxílio da redução de dimensionalidade via PCA. Aprofundamos em métodos de tratamento, como a limpeza dos dados, normalização e redução de dimensionalidade, entre outros.

Também foi realizada uma análise da distribuição dos dados de Y\_treino, pois essa análise inicial das atividades fornece informações fundamentais para uma modelagem mais eficiente, ajudando a identificar problemas nos dados e orientando o processo de construção de modelos preditivos de forma mais eficaz. A seguir, a imagem com o resultado obtido.

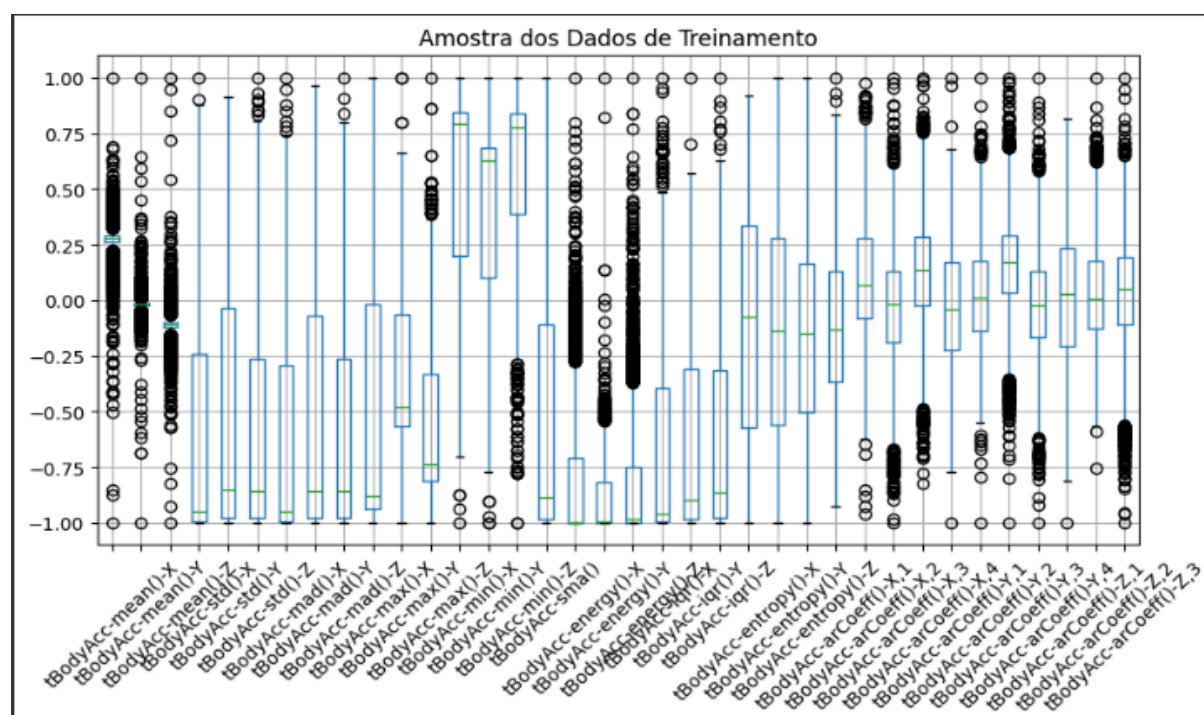
Figura 1 - Distribuição de Y\_treino



Fonte: Elaborada pelos autores (2024)

Após obtermos o resultado da distribuição de Y\_treino, buscamos explorar ainda mais nossos dados para identificar e tratar os outliers em X\_treino, pois sua presença impacta negativamente nos nossos resultados. A imagem a seguir mostra os dados antes do tratamento dos outliers.

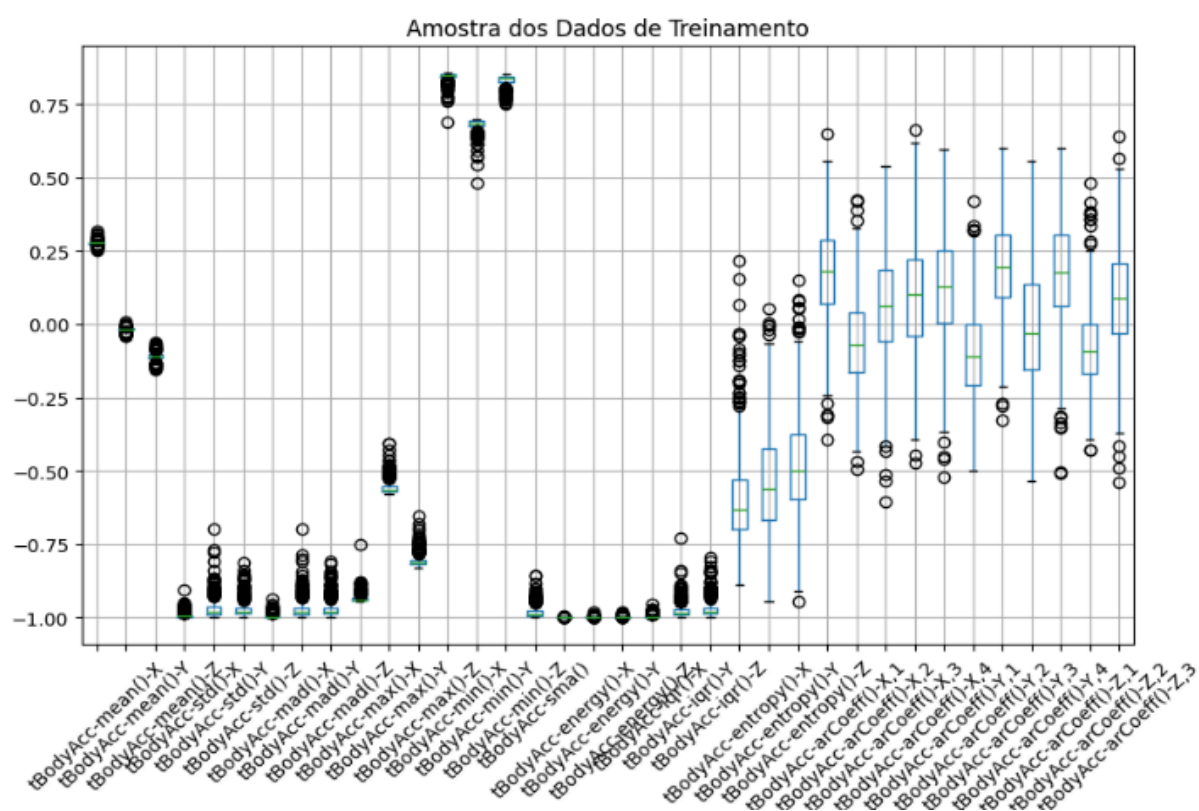
Figura 1 - Amostra de X\_treino



Fonte: Elaborada pelos autores (2024)

Como apresentado na imagem anterior, foi identificada a presença de outliers em nossa base X\_treino. Visando o melhor resultado, realizamos a exclusão desses outliers utilizando o método interquartil (IQR), como podemos visualizar no gráfico seguinte.

Figura 1 - Exclusão de outliers de X\_treino



Fonte: Elaborada pelos autores (2024)

No entanto, apenas a exclusão dos outliers não foi suficiente para obter um bom resultado. Visando esse melhor resultado para nosso K-means, realizamos a normalização desses dados. Como mencionado anteriormente, testamos três métodos de normalização, conforme consta em nosso código, [K-means Geisa e Fabricio](#). A seguir, apresentamos o método selecionado por dupla.

Figura 1 - Robustscaler

```
#SEGUNDO MÉTODO - ROBUSTSCALER
scaler = RobustScaler()
features_scaled = scaler.fit_transform(x_treino_outliers)
x_treino_normatized_roubust = pd.DataFrame(features_scaled)
print(x_treino_normatized_roubust.head())

#Verificando Normalização dos dados com o RobustScaler
amostra_x_treino_normatized_roubust = x_treino_normatized_roubust.columns[:36]
plt.figure(figsize=(12, 6))
x_treino_normatized_roubust[amostra_x_treino_normatized_roubust].boxplot()
plt.title('Amostra dos Dados de Treinamento')
plt.xticks(rotation=45)
plt.show()
```

	0	1	2	3	4	5	6	\
0	0.525725	-0.546882	-0.117525	-0.283168	0.557074	-0.116582	-0.246608	
1	0.261485	-0.704856	-0.365110	-0.724771	-0.169724	-0.811242	-0.739121	
2	0.706240	0.621380	0.216144	-0.437537	-0.287531	-0.501474	-0.486859	
3	-0.084326	-0.208588	0.195065	-0.451161	-0.388968	-0.827685	-0.552524	
4	-1.739440	0.520140	0.202258	0.328218	-0.522804	-0.663193	0.495460	

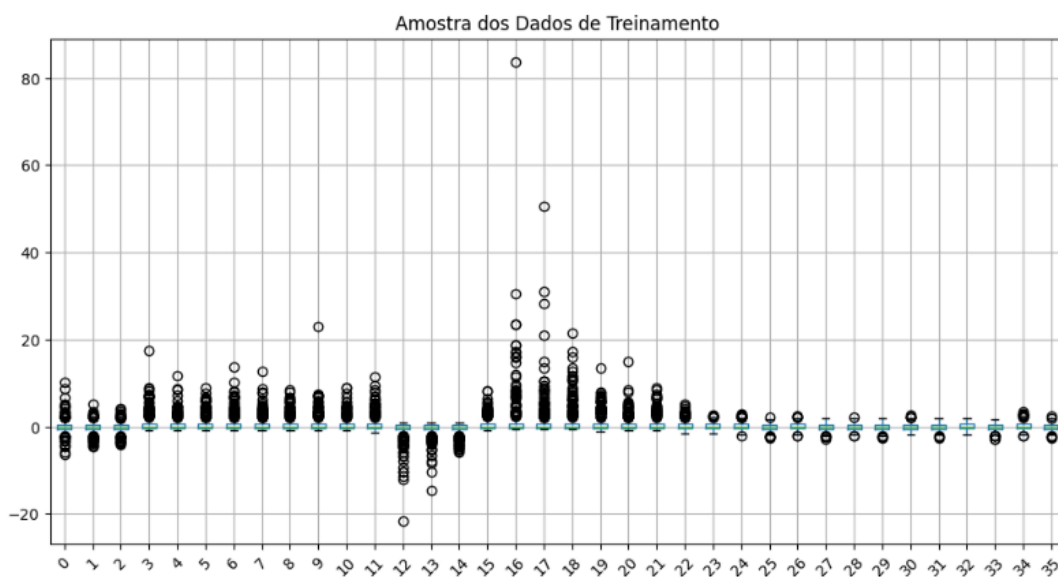
	7	8	9	...	551	552	553	554	\
0	0.629411	-0.110876	-0.202313	...	-0.195378	0.617943	0.827931	-2.186686	
1	-0.121638	-0.760003	-0.558746	...	1.343144	-0.660605	-0.362842	-0.852732	
2	-0.345408	-0.553938	-0.335324	...	0.604266	0.313558	0.705073	-0.367111	
3	-0.421859	-0.774170	-0.261628	...	1.022015	-1.101536	-0.633699	-0.110471	
4	-0.531667	-0.583138	-0.279294	...	-0.453419	0.036030	-0.152322	0.547796	

	555	556	557	558	559	560
0	-0.860955	0.085609	-0.310245	-0.473199	0.008108	-0.351640
1	-0.413919	1.027720	0.928379	-0.464956	0.048055	-0.308777
2	0.254869	-1.176995	0.412270	0.038163	0.609191	0.092602
3	0.455323	0.278263	-1.171708	0.023505	0.595515	0.094576
4	0.975367	0.241034	-0.124701	2.477564	-2.928178	-2.065171

Fonte: Elaborada pelos autores (2024)

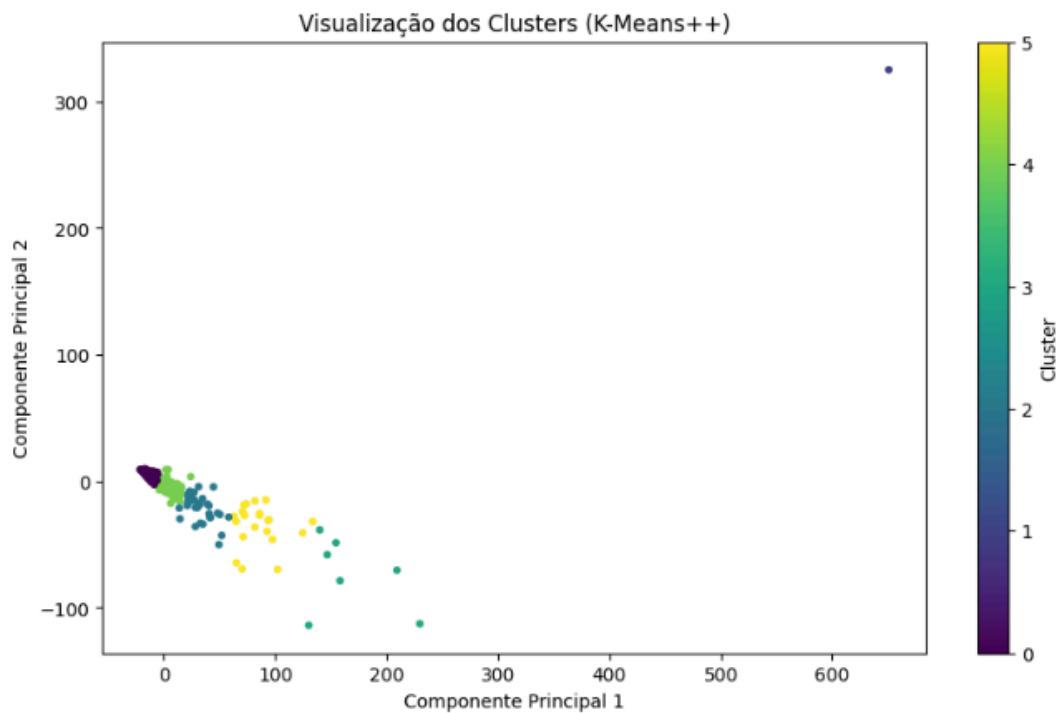
Figura 1 - Normalização realizada



Fonte: Elaborada pelos autores (2024)

Realizamos a redução de dimensionalidade para visualização (2D) e treinamos nosso K-means. Em busca de visualizar o resultado obtido após os métodos, procuramos observar os clusters, como ilustra a imagem a seguir.

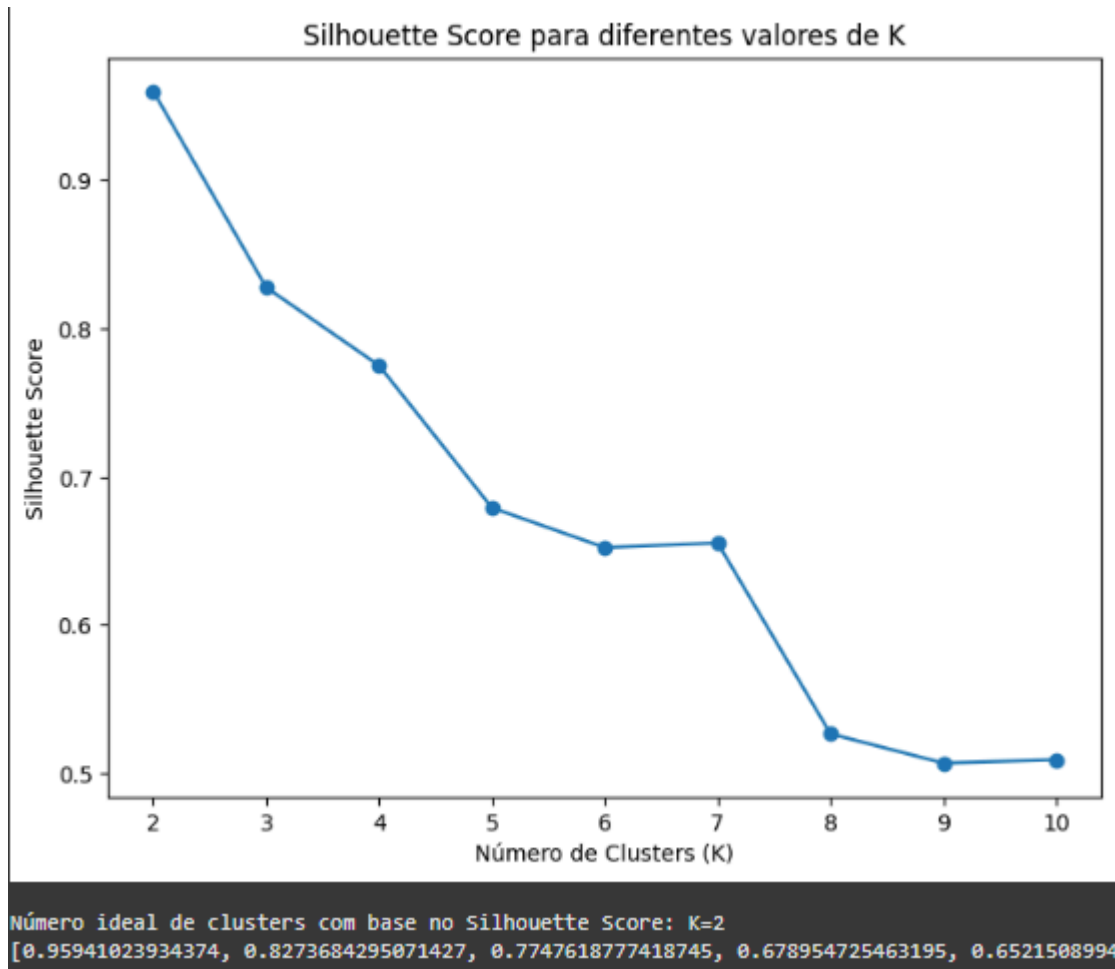
Figura 1 - Visualização de Clusters



Fonte: Elaborada pelos autores (2024)

Diante do resultado obtido com a visualização dos clusters, conforme mostrado na imagem anterior, buscamos analisar, através da Silhouette Score, o nosso K ideal, identificando o número de clusters utilizado em nosso modelo, como apresentado na imagem abaixo.

Figura 1 - Visualização de K



Fonte: Elaborada pelos autores (2024)

Obtivemos aprendizados muito importantes ao longo deste trabalho. Durante esse percurso, enfrentamos desafios, como a presença de dados nulos e outliers, mas, por fim, conseguimos realizar o tratamento de forma eficaz, promovendo, assim, um bom resultado para a performance do modelo, como podemos visualizar nos gráficos a seguir.

Figura 1 - Conversão de valores

```
Silhouette Score após redução de dimensionalidade: 0.9594
```

Fonte: Elaborada pelos autores (2024)



## **Discussão**

Mesmo diante do bom resultado alcançado, nos deparamos com alguns obstáculos ao longo do percurso. Algumas vezes, houve a necessidade de retroceder para, somente depois, conseguir dar continuidade ao tratamento dos dados. Algumas abordagens utilizadas só foram realmente elaboradas após o resultado do treinamento. Enfrentamos resultados muito discrepantes, com muitos ruídos, e fomos obrigados a realizar um novo tratamento dos dados. No entanto, conseguimos implementar métodos que nos trouxeram um resultado eficaz. Uma dessas decisões foi a utilização do método de normalização Robust Scaler, que nos possibilitou obter um melhor resultado final.

## **Conclusão e Trabalhos Futuros**

Este trabalho nos proporcionou uma compreensão mais aprofundada das etapas necessárias para a construção de um modelo mais eficaz. Realizamos uma análise exploratória dos dados, buscando o melhor tratamento, assim como a aplicação de técnicas de normalização para uma melhor interpretação das informações. Abordagens que, sem dúvida, servirão como base para trabalhos futuros.

Novamente, sugerimos para trabalhos posteriores que seria interessante substituir um relatório tão detalhado por uma apresentação em vídeo mais técnica, explicando os métodos e as abordagens utilizadas na elaboração do projeto.

## **Referências**

Reyes-Ortiz, J., Anguita, D., Ghio, A., Oneto, L., & Parra, X. (2013). Human Activity Recognition Using Smartphones [Dataset]. UCI Machine Learning Repository.  
<https://doi.org/10.24432/C54S4K>.

Castanha, R. C. G. (2021). A ciência de dados e a cientista de dados. AtoZ: novas práticas em informação e conhecimento, 10(2), 1 – 4. Recuperado de: <http://dx.doi.org/10.5380/atoz.v10i2.79822>