

**PROJETO:** PREVENDO DIABETES EM MULHERES

**DISCENTE:** GUILHERME ALVES ()

**DISCENTE:** VINICIUS DE SOUZA SANTOS(BI3008061)

**DOCENTE:** DR. MURILO VARGES DA SILVA

**DISCIPLINA:** MDAEL8 – MINERAÇÃO DE DADOS

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

CÂMPUS BIRIGUI

GUILHERME ALVES

VINICIUS DE SOUZA SANTOS

**ESTUDOS DE UMA BASE DADOS PARA PREVER DIABETES USANDO CONCEITOS DE MINERAÇÃO DE DADOS**

Birigui 2022

**GUILHERME ALVES**

**VINICIUS DE SOUZA SANTOS**

**ESTUDOS DE UMA BASE DADOS PARA PREVER DIABETES USANDO CONCEITOS DE MINERAÇÃO DE DADOS**

**Relatório de Trabalho Final Acadêmico apresentado na disciplina de Mineração de dados como requisito para conclusão e aprovação da disciplina.**

Birigui 2023

# LISTA DE FIGURAS

[Figura 1: Base de dados de prevenção de diabete em mulheres 5](#_Toc133860939)

[Figura 2: Saída do DataCleaning do projeto. 8](#_Toc133860940)

[Figura 3: Histograma dos dados de Pressão Arterial 9](#_Toc133860941)

[Figura 4: Histograma dos Dados de IMC 10](#_Toc133860942)

[Figura 5: Saida do algoritmo de Normalização 13](#_Toc133860943)

[Figura 6: Gráficos de Dispersão entre o Resultado e a Espessura da Pele 20](#_Toc133860944)

[Figura 7: Gráficos de setores entre o Resultado e o IMC categorizado 21](#_Toc133860945)

[Figura 8: Gráficos de Histograma entre o resultado e o IMC Categorizado 22](#_Toc133860946)

[Figura 9 Saida do Algoritmo de Medida de Tendencia Central 25](#_Toc133860947)

[Figura 10 Gráfico de Kmeans from scratch 42](#_Toc133860948)

[Figura 11 usando GMM com Decimal Scalling 47](#_Toc133860949)

[Figura 12 Usando GMM com min e max 48](#_Toc133860950)

[Figura 13 usando GMM com Z score 49](#_Toc133860951)

[Figura 14 Resultado do algoritmo de arvore de decisão 54](#_Toc133860952)

[Figura 15 Matriz de Confusão usando KNN e normalizada. 60](#_Toc133860953)

[Figura 16 Matriz de Confusão usando KNN 62](#_Toc133860954)

Sumário

[LISTA DE FIGURAS 2](#_Toc133860918)

[1 Objetivo 3](#_Toc133860919)

[1.1 Sobre a base de dados 3](#_Toc133860920)

[2 Pré-processamento 5](#_Toc133860921)

[3 Normalização e Redução de Dados 11](#_Toc133860922)

[3.1 Normalização dos dados: 11](#_Toc133860923)

[3.2 Redução dos dados: 13](#_Toc133860924)

[4 Análise descritiva de dados – Visualização 17](#_Toc133860925)

[4.1 Gráficos em Histograma, Dispersão e Setores 18](#_Toc133860926)

[5 Análise descritiva de dados – Medidas 23](#_Toc133860927)

[5.1 Medidas de tendência central 24](#_Toc133860928)

[5.2 Medidas de dispersão 25](#_Toc133860929)

[5.3 Medidas de posição relativa 28](#_Toc133860930)

[5.4 Medidas de Associação 31](#_Toc133860931)

[5 Análise de Grupo 33](#_Toc133860932)

[5.1 K-Means 35](#_Toc133860933)

[GMM (Gaussian Mixture Model) 43](#_Toc133860934)

[6 Classificação 50](#_Toc133860935)

[Arvore de Decisão 50](#_Toc133860936)

[KNN (K-Nearest Neighbors) 55](#_Toc133860937)

[Referências Bibliográficas 64](#_Toc133860938)

# 1 Objetivo

Estimular o aluno a aplicar os conhecimentos apresentados no decorrer da disciplina em problemas reais de mineração de dados, utilizando as técnicas de seleção, pré-processamento e transformação de dados, técnicas de visualização de dados, análise descritiva, análise de grupos, classificação e estimação/regressão.

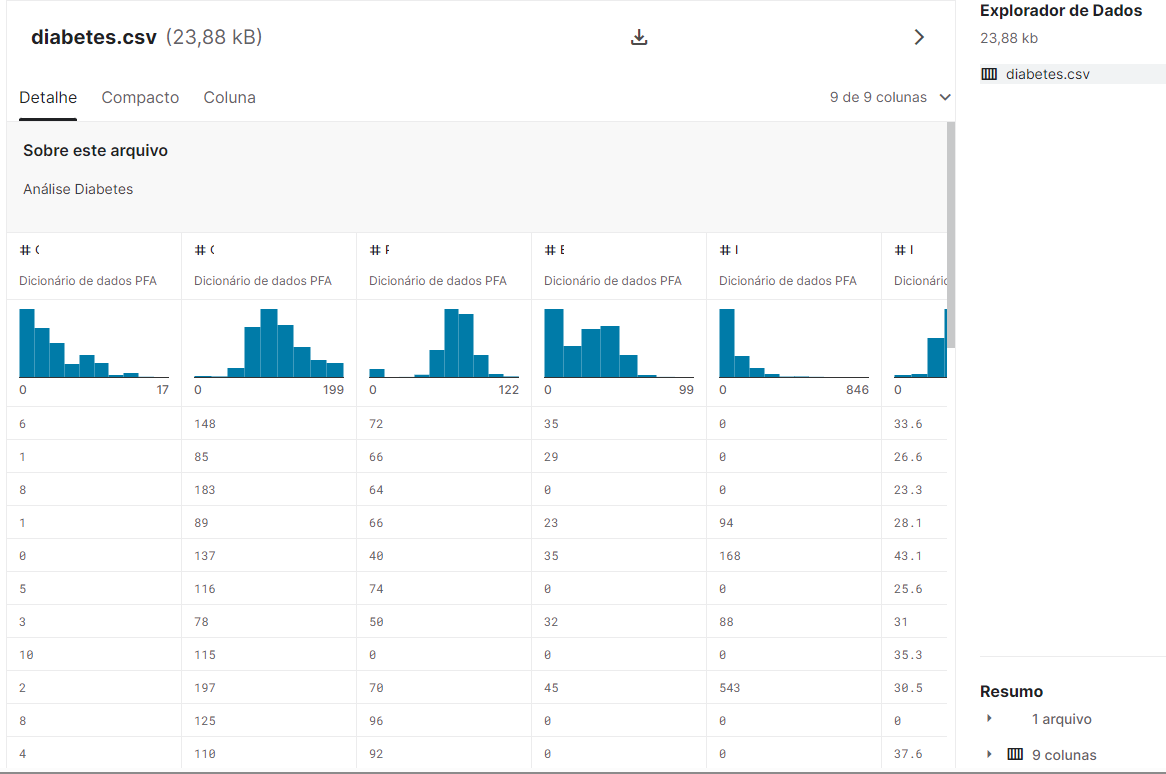
## 1.1 Sobre a base de dados

Este conjunto de dados é originalmente do Instituto Nacional de Diabetes e Doenças Digestivas e Renais. O objetivo do conjunto de dados é prever de forma diagnóstica se um paciente tem diabetes, com base em certas medidas de diagnóstico incluídas no conjunto de dados. Várias restrições foram colocadas na seleção dessas instâncias de um banco de dados maior. Em particular, todos os pacientes aqui são mulheres com pelo menos 21 anos de idade, descendentes dos índios Pima.2 A partir dos dados do arquivo (.csv) Podemos encontrar várias variáveis, algumas delas são independentes (várias variáveis reditivas médicas) e apenas uma variável dependente de destino (resultado).

# 2 Pré-processamento

Nessa primeira etapa do projeto foi feito o pré-processamento dos dados da base de prevenção a diabetes, essa base de dados foi disponibilizada pelo site da Kaggle nela foi abstraída informações relevantes como: Número de gestações, insulina, IMC, espessura da pele entre outros, com essa base de dados vamos fazer a preparação desses dados, conforme observado na figura 1 apresentando a base de dados analisada.

Figura 1: Base de dados de prevenção de diabete em mulheres



fonte: CHAUHAN, 2022

Na aula foi apresentado através de um repositório do professor vários algoritmos de mineração de dados e para esse capítulo usamos apenas o do diretório de processamento.

Para fazer o pré-processamento usamos o algoritmo DataCleaning.py.

import pandas as pd

import numpy as np

def main():

    # Faz a leitura do arquivo

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    output\_file = '0-Datasets/diabetesClear.data'

    input\_file = '0-Datasets/diabetes.data'

    df = pd.read\_csv(input\_file,         # Nome do arquivo com dados

                     names = names,      # Nome das colunas

                     usecols = features, # Define as colunas que serão  utilizadas

                     na\_values='?')      # Define que ? será considerado valores ausentes

    df\_original = df.copy()

    # Imprime as 20 primeiras linhas do arquivo

    print("PRIMEIRAS 20 LINHAS\n")

    print(df.head(20))

    print("\n")

    # Imprime informações sobre dos dados

    print("INFORMAÇÕES GERAIS DOS DADOS\n")

    print(df.info())

    print("\n")

    # Imprime uma analise descritiva sobre dos dados

    print("DESCRIÇÃO DOS DADOS\n")

    print(df.describe())

    print("\n")

    # Imprime a quantidade de valores faltantes por coluna

    print("VALORES FALTANTES\n")

    print(df.isnull().sum())

    print("\n")

    columns\_missing\_value = df.columns[df.isnull().any()]

    print(columns\_missing\_value)

    method = 'mean' # number or median or mean or mode

    for c in columns\_missing\_value:

        UpdateMissingValues(df, c, method)

    print(df.describe())

    print("\n")

    print(df.head(15))

    print(df\_original.head(15))

    print("\n")

    # Salva arquivo com o tratamento para dados faltantes

    df.to\_csv(output\_file, header=False, index=False)

def UpdateMissingValues(df, column, method="mean", number=0):

    if method == 'number':

        # Substituindo valores ausentes por um número

        df[column].fillna(number, inplace=True)

    elif method == 'median':

        # Substituindo valores ausentes pela mediana

        median = df['Density'].median()

        df[column].fillna(median, inplace=True)

    elif method == 'mean':

        # Substituindo valores ausentes pela média

        mean = round(df[column].mean(), 2)  # Modificação na linha da média

        df[column].fillna(mean, inplace=True)

    elif method == 'mode':

        # Substituindo valores ausentes pela moda

        mode = df[column].mode()[0]

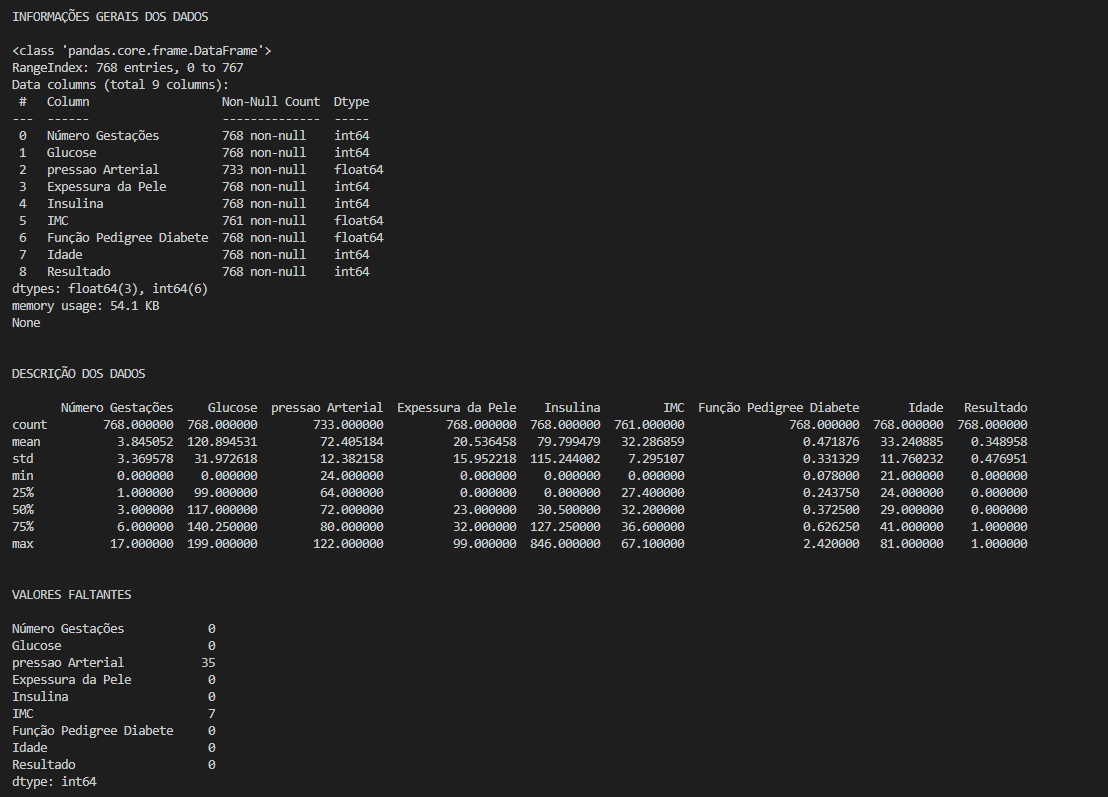
        df[column].fillna(mode, inplace=True)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Ao adaptar esse algoritmo a nossa base de dados obtivemos as seguintes saídas.

Figura 2: Saída do DataCleaning do projeto.



fonte: vscode, 2023

Foi observado duas colunas com dados faltantes e nela foi substituído em uma nova base pela mediana dos valores, foi escolhido a mediada devido os dados da coluna ser assimétrico, ou seja, em um histograma os dados da direita ou esquerda são desiguais conforme verificamos abaixo:

Figura 3: Histograma dos dados de Pressão Arterial

Gráfico, Histograma

Descrição gerada automaticamente

fonte: vscode, 2023

Figura 4: Histograma dos Dados de IMC

Gráfico, Histograma

Descrição gerada automaticamente

fonte: vscode, 2023

# 3 Normalização e Redução de Dados

## 3.1 Normalização dos dados:

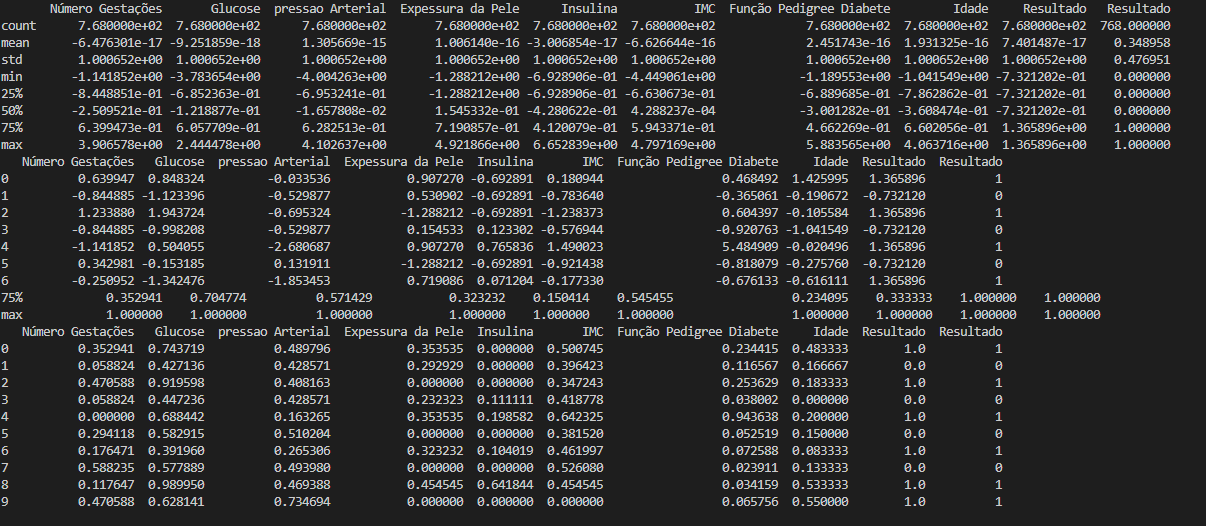
A normalização de dados é o processo de escalar os dados para que todos estejam na mesma escala. Isso é importante porque algumas variáveis podem ter uma escala muito maior do que outras e podem ter um impacto desproporcional na análise. Existem diferentes métodos de normalização, mas um dos mais comuns é a normalização Min-Max, que ajusta os valores para um intervalo entre 0 e 1. Para aplicar a normalização Min-Max à base de dados de prevê diabetes, seguimos esses passos:

* Importar a biblioteca Scikit-learn para a normalização dos dados.
* Dividir a base de dados em duas partes, uma para as variáveis dependentes e outra para a variável de saída.
* Aplicar a normalização Min-Max às variáveis dependentes usando a função MinMaxScaler().
* Concatenar as variáveis dependentes normalizadas com a variável de saída para formar a base de dados normalizada.
* import pandas as pd
* from sklearn.preprocessing import MinMaxScaler
* # Faz a leitura do arquivo
* input\_file = '0-Datasets/diabetesClear.data'
* names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']
* features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']
* target = 'Resultado'
* df = pd.read\_csv(input\_file,    # Nome do arquivo com dados
* names = names) # Nome das colunas
* # Separar as variáveis dependentes da variável de saída
* X = df.drop('Resultado', axis=1)
* y = df['Resultado']
* # Aplicar a normalização Min-Max às variáveis dependentes
* scaler = MinMaxScaler()
* X\_norm = scaler.fit\_transform(X)
* # Criar um novo dataframe com as variáveis dependentes normalizadas e a variável de saída
* df\_norm = pd.DataFrame(X\_norm, columns=X.columns)
* df\_norm['Resultado'] = y
* # Salvar a base de dados normalizada em um novo arquivo csv
* df\_norm.to\_csv('dados\_diabetes\_normalizados.csv', index=False)

O arquivo csv com a base de dados original é lido usando a biblioteca pandas, e as variáveis dependentes (todas as colunas exceto a última) são separadas da variável de saída (a última coluna). Em seguida, a normalização Min-Max é aplicada às variáveis dependentes usando a classe MinMaxScaler da biblioteca Scikit-learn.

O resultado da normalização é armazenado em um novo dataframe chamado df\_norm, que contém as variáveis dependentes normalizadas e a variável de saída original. Finalmente, o dataframe normalizado é salvo em um novo arquivo csv usando a função to\_csv() da biblioteca pandas.

Figura 5: Saida do algoritmo de Normalização



fonte: vscode, 2023

## 3.2 Redução dos dados:

A redução de dados é o processo de reduzir a dimensionalidade dos dados para eliminar variáveis desnecessárias ou redundantes. Isso pode ajudar a reduzir o tempo de processamento e melhorar a precisão do modelo. Uma técnica comum de redução de dados é a Análise de Componentes Principais (PCA), que reduz as variáveis para um número menor de componentes que capturam a maior parte da variação dos dados. Para aplicar a PCA à base de dados fornecida, você pode seguir os seguintes passos:

* Importar a biblioteca Scikit-learn para a redução dos dados.
* Dividir a base de dados em duas partes, uma para as variáveis dependentes e outra para a variável de saída.
* Aplicar a PCA às variáveis dependentes usando a função PCA().
* Selecionar o número de componentes que capturam a maior parte da variação dos dados.
* Transformar as variáveis dependentes reduzidas em uma matriz e concatenar com a variável de saída para formar a base de dados reduzida.
* import pandas as pd
* from sklearn.decomposition import PCA
* # Ler o arquivo csv com a base de dados normalizada
* df\_norm = pd.read\_csv('dados\_diabetes\_normalizados.csv')
* # Separar as variáveis dependentes da variável de saída
* X\_norm = df\_norm.drop('Resultado', axis=1)
* y\_norm = df\_norm['Resultado']
* # Aplicar a PCA às variáveis dependentes normalizadas
* pca = PCA(n\_components=3)
* X\_pca = pca.fit\_transform(X\_norm)
* # Criar um novo dataframe com as variáveis dependentes reduzidas e a variável de saída
* df\_pca = pd.DataFrame(X\_pca, columns=['PCA1', 'PCA2', 'PCA3'])
* df\_pca['Resultado'] = y\_norm
* # Salvar a base de dados reduzida em um novo arquivo csv
* df\_pca.to\_csv('dados\_diabetes\_reduzidos.csv', index=False)

O arquivo csv com a base de dados normalizada é lido usando a biblioteca pandas, e as variáveis dependentes (todas as colunas exceto a última) são separadas da variável de saída (a última coluna).

Em seguida, a PCA é aplicada às variáveis dependentes normalizadas usando a classe PCA da biblioteca Scikit-learn, especificando o número de componentes que se deseja manter. Neste exemplo, estamos mantendo os três primeiros componentes principais.

O resultado da redução é armazenado em um novo dataframe chamado df\_pca, que contém as variáveis dependentes reduzidas e a variável de saída original. Finalmente, o dataframe reduzido é salvo em um novo arquivo csv usando a função to\_csv() da biblioteca pandas.

Logo abaixo temos o resultado da saída do PCA e percebemos algo positivo que os dados estão bastante dispersos uns dos outros.

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

Em resumo, a normalização e redução de dados são técnicas importantes para pré-processar a base de dados utilizada para diabetes antes de aplicar algoritmos para prever diabetes. A normalização ajuda a garantir que todas as variáveis estejam na mesma escala, enquanto a redução ajuda a reduzir a dimensionalidade dos dados para melhorar a precisão do modelo.

# 4 Análise descritiva de dados – Visualização

A análise descritiva de dados é uma técnica muito utilizada em mineração de dados, que tem como objetivo analisar e entender os dados de um conjunto de dados para obter informações relevantes. Essa técnica permite entender melhor as características dos dados, identificar padrões, tendências e anomalias, além de verificar a qualidade dos dados.

A visualização de dados é uma ferramenta essencial para a análise descritiva de dados, pois permite a representação gráfica dos dados de uma forma que facilita a compreensão e a interpretação das informações contidas no conjunto de dados. Com a visualização, é possível identificar padrões e tendências que não seriam detectados com apenas a análise dos dados em formato tabular.

Entre as principais técnicas de visualização utilizadas na análise descritiva de dados estão o histograma, o gráfico de setores, o gráfico de barras, o gráfico de dispersão e a matriz de correlação. Cada uma dessas técnicas é utilizada para representar diferentes tipos de dados e para identificar diferentes tipos de padrões e tendências.

Em resumo, a análise descritiva de dados e a visualização de dados são técnicas fundamentais em mineração de dados para compreender e extrair informações úteis de grandes conjuntos de dados.

## 4.1 Gráficos em Histograma, Dispersão e Setores

Os gráficos de histograma, dispersão e setores são comumente utilizados na análise descritiva de dados por suas capacidades de visualização eficaz e fácil interpretação dos dados.

O gráfico de histograma é útil para visualizar a distribuição dos dados e pode ajudar a identificar padrões, como picos e caudas longas, bem como a presença de valores atípicos. É particularmente útil quando se trabalha com dados contínuos ou discretos.

O gráfico de dispersão é utilizado para analisar a relação entre duas variáveis. É capaz de identificar padrões, correlações e relacionamentos entre variáveis e pode ser usado para identificar possíveis causas e efeitos.

Já o gráfico de setores é comumente usado para mostrar a proporção de cada categoria em um conjunto de dados. É útil quando se trabalha com dados categóricos e pode ajudar a identificar padrões ou desequilíbrios entre as categorias.

Esses gráficos são considerados eficazes porque são fáceis de entender e interpretar. Eles permitem que os usuários vejam rapidamente as tendências, padrões e correlações nos dados, sem precisar olhar para tabelas de números ou dados brutos. Além disso, esses gráficos podem ser usados para identificar problemas nos dados, como valores discrepantes ou ausentes, o que pode levar a análises mais precisas e confiáveis.

Em resumo, esses gráficos são valiosos para a análise descritiva de dados porque oferecem uma maneira visualmente atraente e intuitiva de explorar os dados e comunicar os resultados de maneira clara e concisa.

Abaixo segue o algoritmo em Python utilizado para a plotagem desses gráficos conforme a base de dados de Prever Diabetes.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('0-Datasets/diabetesClear.data')

for coluna in df.columns:

    # Histograma

    plt.figure()

    sns.histplot(data=df, x=coluna)

    plt.title(f'Histograma da coluna {coluna}')

    plt.savefig(f'3-Analise Descritiva - Visualização/Graficos/Histograma/historgrama\_{coluna}.png')

    plt.close()

    # Gráfico de setores

    plt.figure()

    df[coluna].value\_counts().plot(kind='pie')

    plt.title(f'Gráfico de setores da coluna {coluna}')

    plt.savefig(f'3-Analise Descritiva - Visualização/Graficos/Grafico de Setores/grafico\_setores\_{coluna}.png')

    plt.close()

    # Dispersão

    plt.figure()

    sns.scatterplot(data=df, x=coluna, y='Outcome')

    plt.title(f'Dispersão da coluna {coluna} em relação à coluna alvo')

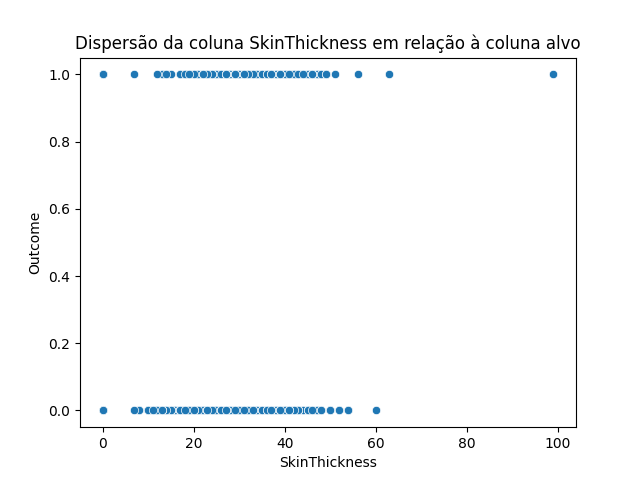
    plt.savefig(f'3-Analise Descritiva - Visualização/Graficos/Dispersao/dispersao\_{coluna}.png')

    plt.close()

Nesse código, vemos todas correlação além da coluna de resultado com as demais temos uma relação de todas as colunas entre sim e podemos ver mais detalhadamente a relação de cada coluna com a outra, ao total foi gerado 71 gráficos pois temos pois temos 8 colunas em 8 sendo relaciona e por último cada coluna relacionada com nossa coluna alvo e com esses gráficos conseguimos ter um entendimento melhor

Abaixo temos alguns exemplos de saída desses gráficos em histograma.

Figura 6: Gráficos de Dispersão entre o Resultado e a Espessura da Pele



fonte: elaborado pelo autor (2023)

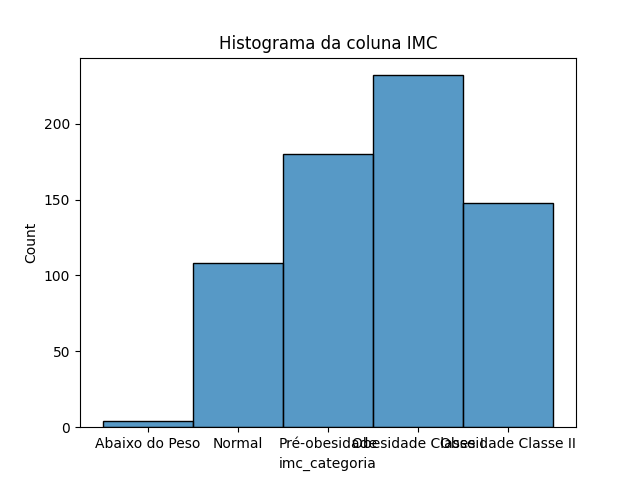
Figura 7: Gráficos de setores entre o Resultado e o IMC categorizado

Gráfico, Gráfico de pizza

Descrição gerada automaticamente

fonte: elaborado pelo autor (2023)

Figura 8: Gráficos de Histograma entre o resultado e o IMC Categorizado



fonte: elaborado pelo autor (2023)

# 5 Análise descritiva de dados – Medidas

Análise descritiva de dados é uma técnica de mineração de dados que busca descrever as características de um conjunto de dados, sem inferir relações causais ou preditivas. Ela é utilizada para explorar os dados, obter insights e compreender melhor as informações disponíveis.

As medidas estatísticas são ferramentas fundamentais na análise descritiva de dados. As medidas de tendência central, dispersão, posição relativa e associação são exemplos de medidas estatísticas utilizadas na análise descritiva de dados.

Medidas de tendência central: São medidas que indicam o ponto central de um conjunto de dados. As três medidas de tendência central mais comuns são a média, a mediana e a moda.

Medidas de dispersão: São medidas que indicam o grau de variação dos dados em relação a uma medida central. As medidas de dispersão mais comuns são o desvio padrão, a variância e o coeficiente de variação.

Medidas de posição relativa: São medidas que indicam a posição de um valor em relação aos demais valores de um conjunto de dados. As medidas de posição relativa mais comuns são o percentil e o quartil.

Medidas de associação: São medidas que indicam a relação entre duas ou mais variáveis. As medidas de associação mais comuns são a correlação e a covariância.

Em resumo, a análise descritiva de dados utiliza medidas estatísticas para resumir e descrever os dados, de forma a obter insights e compreender melhor as informações disponíveis. As medidas de tendência central, dispersão, posição relativa e associação são algumas das medidas estatísticas mais comuns utilizadas na análise descritiva de dados.

## 5.1 Medidas de tendência central

Medidas de tendência central são utilizadas para resumir a distribuição dos dados de uma variável quantitativa, indicando onde a maioria dos valores estão concentrados. As três medidas de tendência central mais comuns são a média, a mediana e a moda.

A média é a soma de todos os valores dividida pelo número total de valores e é frequentemente utilizada como uma medida padrão de tendência central. A mediana é o valor que divide a distribuição ao meio, de modo que metade dos valores estão acima dela e metade estão abaixo. Já a moda é o valor mais frequente na distribuição, ou seja, o valor que aparece com maior frequência.

Cada uma dessas medidas de tendência central pode ser útil em diferentes situações. A média é sensível a valores extremos e pode não ser representativa de uma distribuição assimétrica ou com outliers. A mediana é uma medida mais robusta que a média, pois não é afetada por valores extremos e é frequentemente utilizada quando a distribuição não é normal. Já a moda é útil para identificar o valor mais comum em uma distribuição e é frequentemente usada em distribuições simétricas ou com uma única moda clara.

Em resumo, as medidas de tendência central são uma maneira de resumir e descrever a distribuição dos dados de uma variável quantitativa, ajudando a entender onde a maioria dos valores estão concentrados e como se comportam em relação aos outros valores.

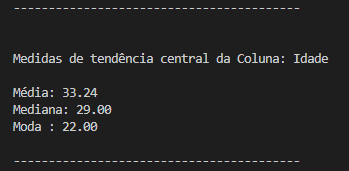
    # Medidas de tendência central

    media = df[target].mean()

    mediana = df[target].median()

    moda = df[target].mode()[0]

Figura 9 Saida do Algoritmo de Medida de Tendencia Central



Oque podemos observar nessa saída e que média é de 33.24 anos, mas a mediana é de apenas 29 anos, o que sugere que há alguns pacientes com idades mais altas que estão aumentando a média. A moda de 22 anos indica que este é o valor mais comum para a idade dos pacientes na base de dados.

## 5.2 Medidas de dispersão

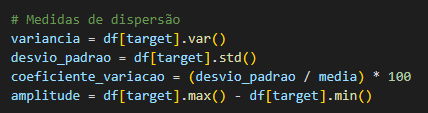
As medidas de dispersão, em mineração de dados, são usadas para descrever a variabilidade ou a dispersão dos dados de uma variável quantitativa em torno de uma medida de tendência central, como a média ou a mediana.

As medidas de dispersão mais comuns incluem a amplitude, a variância e o desvio padrão. A amplitude é a diferença entre o maior e o menor valor na distribuição. Já a variância é uma medida de quanto os valores variam em relação à média, calculada como a média dos quadrados das diferenças entre cada valor e a média. O desvio padrão é a raiz quadrada da variância e indica quanto os valores tendem a se desviar da média.

As medidas de dispersão são importantes porque fornecem informações sobre a variabilidade dos dados em torno da medida de tendência central. Uma distribuição com baixa dispersão terá a maioria dos valores próximos à medida de tendência central, enquanto uma distribuição com alta dispersão terá valores mais espalhados ao redor da medida de tendência central.

Assim como as medidas de tendência central, as medidas de dispersão podem ser úteis em diferentes situações. A amplitude é uma medida simples e direta de dispersão, mas pode ser influenciada por valores extremos. A variância e o desvio padrão são mais robustos, mas podem ser difíceis de interpretar, especialmente em unidades não familiares.

Em resumo, as medidas de dispersão são usadas para descrever a variabilidade dos dados em torno de uma medida de tendência central, e são importantes para entender a forma e a dispersão da distribuição dos dados.



Texto

Descrição gerada automaticamente

Os resultados das medidas de dispersão indicam que há uma variação considerável na idade dos pacientes na base de dados de Prever Diabetes. A variância de 138.30 indica que os valores de idade estão espalhados em torno da média de 33.24 anos, enquanto o desvio padrão de 11.76 indica que há uma variação relativamente alta em relação à média. Isso sugere que a idade dos pacientes está distribuída de forma mais ampla do que seria esperado se fosse distribuída normalmente.

O coeficiente de variação de 35.38% indica que a variação relativa na idade dos pacientes na base de dados é relativamente alta em relação à média, o que sugere que a idade dos pacientes varia bastante. A amplitude de 60.00 indica que a diferença entre a idade mais jovem e a mais velha na base de dados é de 60 anos, o que indica uma grande variabilidade na idade dos pacientes.

Esses resultados indicam que a base de dados de Prever Diabetes pode ter uma distribuição não normal em relação à idade dos pacientes, com uma variação considerável em torno da média. Isso sugere que a idade dos pacientes pode ser um fator importante a ser considerado ao prever o diabetes nessa base de dados, uma vez que há uma grande variação na idade dos pacientes.

Os resultados das medidas de tendência central e de dispersão da idade dos pacientes na base de dados de Prever Diabetes são importantes para entender a distribuição da idade e a variação dos valores na amostra. Esses resultados podem afetar a análise da base de dados, dependendo do objetivo da análise.

Por exemplo, se o objetivo da análise for avaliar a associação entre a idade dos pacientes e a ocorrência de diabetes, a grande variação na idade pode indicar a necessidade de uma análise mais detalhada, como o uso de técnicas de estratificação por idade, para avaliar como diferentes faixas etárias podem estar associadas a diferentes riscos de diabetes.

Além disso, a distribuição não normal dos valores de idade pode afetar a seleção e a validação de modelos de previsão de diabetes, pois os modelos de regressão e classificação geralmente assumem uma distribuição normal das variáveis de entrada. É importante considerar esses fatores ao interpretar os resultados da base de dados e ao selecionar as técnicas de análise mais adequadas para alcançar os objetivos da análise.

## 5.3 Medidas de posição relativa

As medidas de posição relativa, em mineração de dados, são usadas para descrever a posição de um valor dentro de uma distribuição em relação a outras medidas, como a média ou a mediana. Essas medidas fornecem informações sobre a posição relativa de um valor em relação à distribuição como um todo.

As medidas de posição relativa mais comuns são o percentil e o quartil. O percentil é o valor abaixo do qual uma porcentagem especificada de observações em uma distribuição está contida. Por exemplo, o percentil 50 é a mediana, ou o valor abaixo do qual 50% dos valores estão contidos. Já os quartis dividem a distribuição em quartos iguais, com o primeiro quartil (Q1) sendo o valor abaixo do qual 25% dos valores estão contidos, o segundo quartil (Q2) sendo a mediana e o terceiro quartil (Q3) sendo o valor abaixo do qual 75% dos valores estão contidos.

As medidas de posição relativa são úteis para comparar valores individuais com a distribuição como um todo e avaliar a posição relativa de um valor em relação aos outros valores. Por exemplo, um valor acima do terceiro quartil indica que está acima da maioria dos valores na distribuição, enquanto um valor abaixo do primeiro quartil indica que está abaixo da maioria dos valores.

Em resumo, as medidas de posição relativa são usadas para descrever a posição de um valor em relação à distribuição como um todo, e são importantes para entender a posição relativa de um valor em relação aos outros valores. As medidas de posição relativa podem ser usadas em conjunto com as medidas de tendência central e dispersão para fornecer uma descrição completa da distribuição dos dados.

    print(f"\nMedidas de posição relativa da Coluna: {target}\n")

    print('Z Score:\n{}\n'.format((df[target] - df[target].mean())/df[target].std())) # Z Score

    print("Primeiro quartil: {:.2f}".format(Q1))

    print("Segundo quartil (Mediana): {:.2f}".format(Q2))

    print("Terceiro quartil: {:.2f}".format(Q3))

    print("\n-----------------------------------------\n")

Texto

Descrição gerada automaticamente

Os resultados das medidas de posição relativa, como o Z-score e os quartis, fornecem informações importantes sobre como a idade dos pacientes na base de dados está distribuída em relação à média e aos limites inferiores e superiores.

O Z-score indica quantos desvios padrão cada valor de idade está acima ou abaixo da média da amostra. Por exemplo, um Z-score de 1.4 indica que um paciente tem uma idade 1.4 desvios padrão acima da média da amostra. Isso pode ser útil para identificar valores extremos na amostra, que podem afetar a análise da base de dados.

Os quartis fornecem informações sobre a distribuição da idade dos pacientes na base de dados. O primeiro quartil (Q1) de 24 anos indica que 25% dos pacientes têm uma idade menor ou igual a 24 anos. O segundo quartil (Q2), que é a mediana, de 29 anos indica que 50% dos pacientes têm uma idade menor ou igual a 29 anos. O terceiro quartil (Q3) de 41 anos indica que 75% dos pacientes têm uma idade menor ou igual a 41 anos. Isso sugere que a maioria dos pacientes na base de dados tem idades abaixo de 41 anos, mas há alguns pacientes mais velhos que estão aumentando a média.

Esses resultados indicam que a idade dos pacientes na base de dados de Prever Diabetes está distribuída de forma bastante ampla, com valores extremos que podem afetar a análise. Além disso, a maioria dos pacientes tem idades abaixo de 41 anos, o que pode ser útil para identificar grupos de pacientes com idades semelhantes que possam ter riscos de diabetes semelhantes.

## 5.4 Medidas de Associação

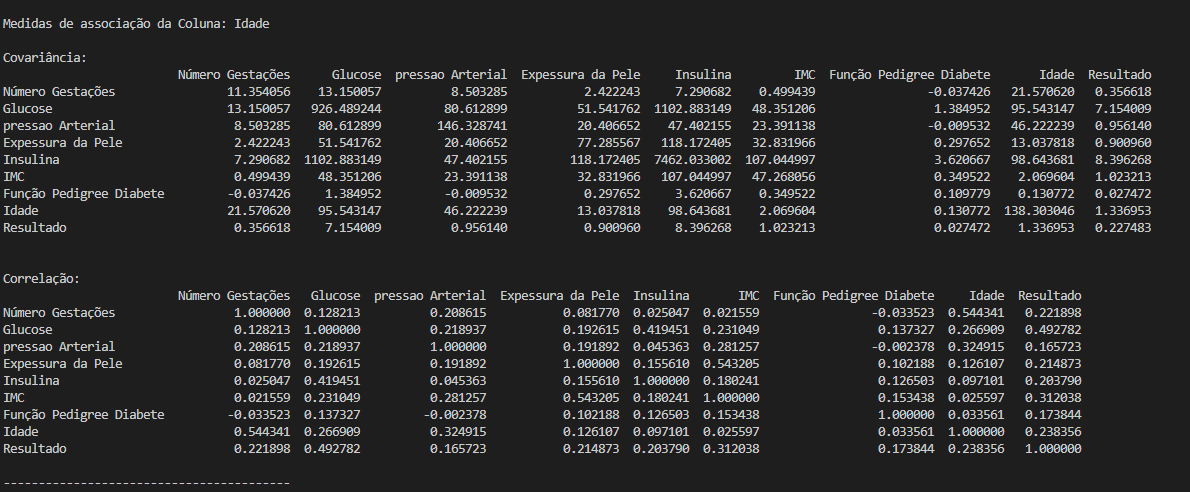
As medidas de associação, em mineração de dados, são usadas para avaliar a relação entre duas variáveis. Elas fornecem informações sobre a direção e a força da associação entre as variáveis, e são usadas para identificar padrões e relações nos dados.

As medidas de associação mais comuns incluem a correlação e a covariância. A correlação mede a força da relação linear entre duas variáveis, variando de -1 (correlação negativa perfeita) a +1 (correlação positiva perfeita), com 0 indicando ausência de correlação. A covariância mede a relação entre duas variáveis, mas é afetada pela escala das variáveis e pode ser difícil de interpretar.

As medidas de associação são importantes porque ajudam a entender como as variáveis se relacionam entre si. Por exemplo, a correlação pode ser usada para determinar se existe uma relação positiva ou negativa entre duas variáveis, e se essa relação é forte ou fraca.

No entanto, é importante ter cuidado ao interpretar as medidas de associação, pois elas podem indicar uma relação aparente entre duas variáveis, mas não necessariamente uma relação causal. Além disso, outras variáveis ou fatores podem influenciar a associação observada entre as variáveis, o que pode levar a interpretações equivocadas.

Em resumo, as medidas de associação são usadas para avaliar a relação entre duas variáveis e fornecer informações sobre a direção e a força da associação. Elas são úteis para identificar padrões e relações nos dados, mas devem ser interpretadas com cuidado e considerando outros fatores que possam influenciar a associação observada.



Com base nas medidas de associação da coluna "Idade", podemos notar que a covariância entre a idade e o número de gestações é de 21.57, indicando uma relação positiva, ou seja, quanto maior a idade, maior o número de gestações.

Além disso, a correlação entre idade e resultado do teste de diabetes é de 0,238, indicando uma relação positiva fraca, ou seja, há uma leve tendência de que pessoas mais velhas tenham maior probabilidade de apresentar resultado positivo no teste de diabetes.

No entanto, é importante ressaltar que a idade é apenas uma das variáveis que podem estar relacionadas ao resultado do teste de diabetes, e é necessário analisar outras variáveis para uma avaliação mais completa.

É possível observar que a idade tem correlação positiva com o resultado do teste de diabetes, indicando que pessoas mais velhas tendem a ter maior probabilidade de ter diabetes. Além disso, a idade tem correlação moderada com o número de gestações e forte correlação com a idade gestacional. No entanto, para uma análise mais detalhada, é necessário considerar os demais resultados das medidas de associação e aprofundar no contexto e na interpretação clínica dos dados.

# 5 Análise de Grupo

As análises de grupos, também conhecidas como análises de clusters, são técnicas de mineração de dados que têm como objetivo encontrar grupos de objetos similares em um conjunto de dados.

A fundamentação teórica das análises de grupos está relacionada com técnicas de aprendizado não supervisionado, que buscam identificar estruturas subjacentes em dados sem a necessidade de um conjunto de dados rotulados.

Existem diferentes métodos de análises de grupos, como a análise hierárquica, a análise k-means e a análise de mistura de Gaussianas. Em geral, esses métodos utilizam uma medida de distância para calcular a similaridade entre objetos e, a partir disso, agrupá-los em clusters.

A escolha da medida de distância depende do tipo de dados sendo analisados e da natureza do problema em questão. Além disso, é importante definir critérios para avaliar a qualidade dos clusters obtidos, como a coerência interna dos grupos e a separação entre eles.

As análises de grupos são amplamente utilizadas em diversas áreas, como biologia, marketing, finanças e ciência da computação, para identificar padrões e estruturas em dados não rotulados, possibilitando a geração de insights e tomada de decisões.

Existem diversas ferramentas e técnicas que podem ser utilizadas para realizar análises de grupos. Algumas das principais são:

Análise hierárquica: essa técnica permite criar uma hierarquia de clusters, na qual grupos menores são agrupados em grupos maiores, e assim por diante. Existem dois tipos principais de análise hierárquica: aglomerativa e divisiva.

Análise k-means: essa técnica agrupa os dados em k clusters, em que k é definido pelo usuário. O algoritmo calcula centróides para cada grupo e ajusta os clusters de forma iterativa, até que a convergência seja alcançada.

Análise de mistura de Gaussianas: essa técnica é utilizada para modelar dados em que a distribuição dos grupos é gaussiana. O algoritmo estima os parâmetros da distribuição de cada grupo, como a média e a variância, e utiliza esses parâmetros para atribuir objetos a clusters.

Análise de densidade: essa técnica utiliza uma medida de densidade para encontrar regiões de alta densidade em um conjunto de dados. Os objetos dentro dessas regiões são agrupados em clusters.

Redução de dimensionalidade: essa técnica é utilizada para reduzir a dimensionalidade do conjunto de dados, de forma a facilitar a análise. Algumas técnicas comuns de redução de dimensionalidade incluem a Análise de Componentes Principais (PCA) e a Análise de Fatores.

Além dessas técnicas, existem diversas ferramentas e softwares que podem ser utilizados para realizar análises de grupos, como o Python com as bibliotecas Scikit-Learn e Pandas, o R com os pacotes cluster e factoextra, e o Weka. A escolha da ferramenta depende das necessidades do projeto e da familiaridade do usuário com cada ferramenta.

## 5.1 K-Means

O método K-means é uma técnica de análise de grupos amplamente utilizada em mineração de dados e aprendizado de máquina. Ele é utilizado para agrupar um conjunto de dados em K clusters, em que K é um número pré-definido de grupos.

O algoritmo K-means funciona de forma iterativa e envolve os seguintes passos:

Inicialização: o número K de clusters é definido pelo usuário e o algoritmo seleciona aleatoriamente K objetos do conjunto de dados como centróides iniciais.

Atribuição de objetos a clusters: cada objeto do conjunto de dados é atribuído ao cluster cujo centróide está mais próximo. A distância entre os objetos e os centróides é calculada utilizando uma medida de distância, geralmente a distância euclidiana.

Atualização dos centróides: os centróides de cada cluster são atualizados para refletir a posição dos objetos que foram atribuídos a ele.

Repetição: os passos 2 e 3 são repetidos até que os centróides deixem de se mover ou até que o número máximo de iterações seja alcançado.

Convergência: quando a iteração termina, o algoritmo tem como resultado um conjunto de K clusters, onde cada objeto pertence a um dos clusters.

Uma das principais vantagens do método K-means é a sua eficiência computacional, sendo capaz de lidar com grandes conjuntos de dados. No entanto, a qualidade dos clusters obtidos pode ser afetada pela escolha dos centróides iniciais, que pode levar a soluções subótimas. Para lidar com esse problema, é comum executar o algoritmo várias vezes com diferentes valores iniciais de centróides e escolher a solução com o menor valor de função objetivo.

O método K-means é amplamente utilizado em diversas áreas, como marketing, biologia, finanças e ciência da computação, para segmentar clientes, identificar grupos de proteínas com funções semelhantes, agrupar empresas com perfis financeiros semelhantes, entre outras aplicações.

Para determinar o número ideal de clusters, podemos usar uma técnica chamada "método do cotovelo" (elbow method) que consiste em executar o algoritmo KMeans para diferentes valores de k e plotar o valor da métrica de inércia (Soma dos Quadrados das Distâncias dentro do Cluster) em relação ao número de clusters. O objetivo é escolher o valor de k que apresenta a maior redução na inércia, que é geralmente identificado como o ponto de inflexão na curva.

Para implementar o método do cotovelo em seu código, você pode adicionar o seguinte trecho de código após a definição da função "KMeans\_scratch":

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

inertias = []

for k in range(1, 11):

kmeans = KMeans\_scratch(k=k, max\_iter=100, tol=0.001)

kmeans.fit(X\_pca)

inertias.append(kmeans.inertia\_)

plt.plot(range(1, 11), inertias)

plt.title('Método do Cotovelo')

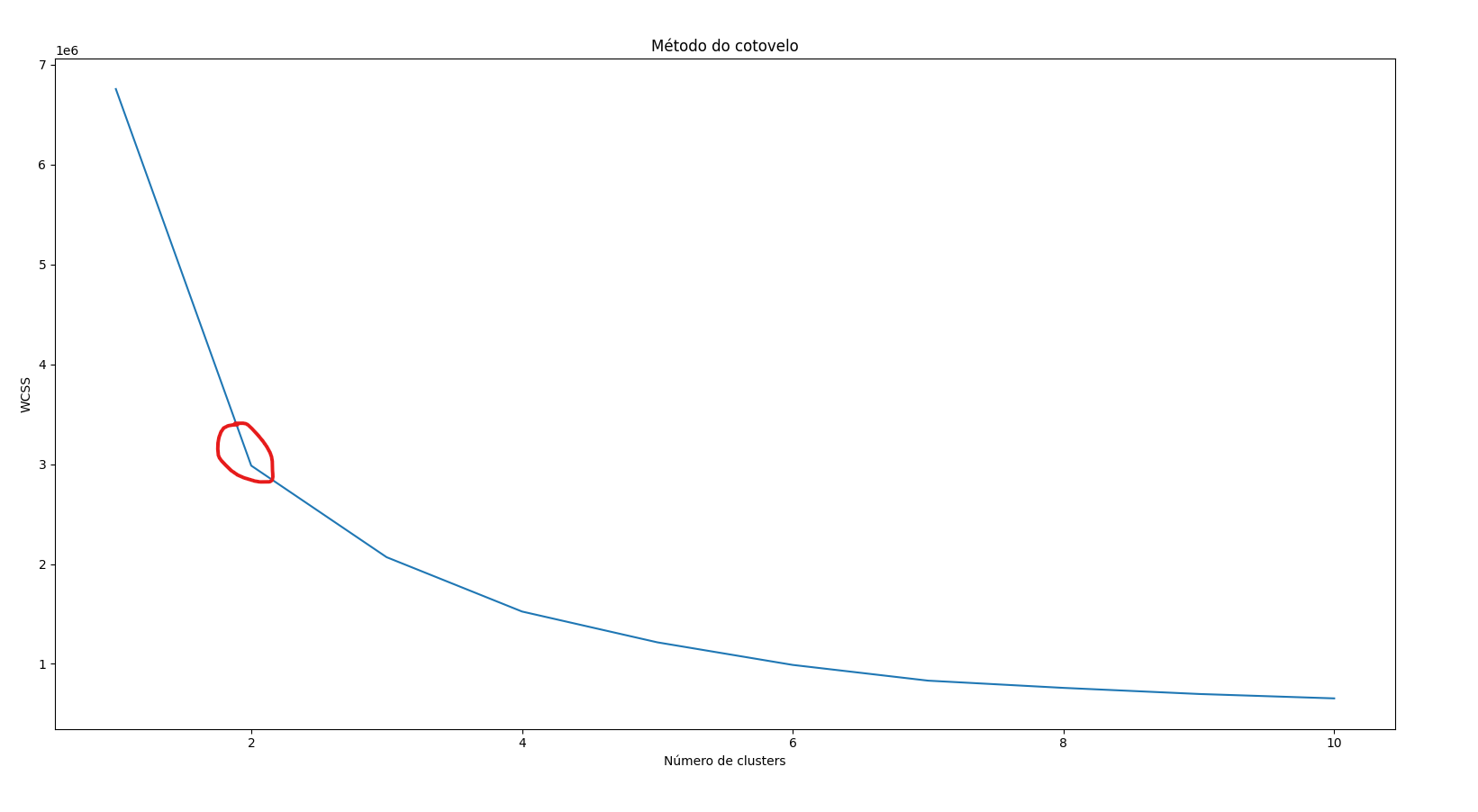
plt.xlabel('Número de Clusters')

plt.ylabel('Inércia')

plt.show()

Esse código executa o algoritmo KMeans para k variando de 1 a 10, armazena a inércia de cada modelo gerado em uma lista e plota a curva da inércia em relação ao número de clusters. O objetivo é identificar o ponto de inflexão na curva e escolher o valor de k que apresenta a maior redução na inércia.

Após executar o código acima, você pode analisar o gráfico gerado e identificar o ponto de inflexão na curva. Esse ponto representa o número ideal de clusters para o seu conjunto de dados.



fonte: elaborado pelo autor (2023)

Com esse resultamos concluímos que o número de Clusters ideal para essa base seria 2. Com esse entendimento podemos gerar os gráficos do Kmeans pelo Código abaixo.

*#Implementation of Kmeans from scratch and using sklearn*

*#Loading the required modules*

import numpy as np

from scipy.spatial.distance import cdist

from sklearn.datasets import load\_digits

from sklearn.decomposition import PCA

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.metrics import silhouette\_samples

import matplotlib.pyplot as plt

import pandas as pd

*#Defining our kmeans function from scratch*

def KMeans\_scratch(*x*,*k*, *no\_of\_iterations*):

    idx = np.random.choice(len(*x*), *k*, *replace*=False)

*#Randomly choosing Centroids*

    centroids = *x*[idx, :] *#Step 1*

*#finding the distance between centroids and all the data points*

    distances = cdist(*x*, centroids ,'euclidean') *#Step 2*

*#Centroid with the minimum Distance*

    points = np.array([np.argmin(i) for i in distances]) *#Step 3*

*#Repeating the above steps for a defined number of iterations*

*#Step 4*

    for \_ in range(*no\_of\_iterations*):

        centroids = []

        for idx in range(*k*):

*#Updating Centroids by taking mean of Cluster it belongs to*

            temp\_cent = *x*[points==idx].mean(*axis*=0)

            centroids.append(temp\_cent)

        centroids = np.vstack(centroids) *#Updated Centroids*

        distances = cdist(*x*, centroids ,'euclidean')

        points = np.array([np.argmin(i) for i in distances])

    return points

def show\_digitsdataset(*digits*):

    fig = plt.figure(*figsize*=(6, 6))  *# figure size in inches*

    fig.subplots\_adjust(*left*=0, *right*=1, *bottom*=0, *top*=1, *hspace*=0.05, *wspace*=0.05)

    for i in range(64):

        ax = fig.add\_subplot(8, 8, i + 1, *xticks*=[], *yticks*=[])

*# ax.imshow(digits.axes[i], cmap=plt.cm.binary, interpolation='nearest')*

*# label the image with the target value*

        ax.text(0, 7, str(*digits*.resultado[i]))

*#fig.show()*

def plot\_samples(*projected*, *labels*, *title*):

    fig = plt.figure()

    u\_labels = np.unique(*labels*)

    for i in u\_labels:

        plt.scatter(*projected*[*labels* == i , 0] , *projected*[*labels* == i , 1] , *label* = i,

*edgecolor*='none', *alpha*=0.5, *cmap*=plt.cm.get\_cmap('tab10', 10))

    plt.xlabel('component 1')

    plt.ylabel('component 2')

    plt.legend()

    plt.title(*title*)

def main():

    input\_file = '0-Datasets/diabetesClear.data'

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    target = 'Resultado'

    digits = pd.read\_csv(input\_file,         *# Nome do arquivo com dados #df =  data framing*

*names* = names,      *# Nome das colunas*

*usecols* = features, *# Define as colunas que serão  utilizadas*

*na\_values*='?')      *# Define que ? será considerado valores ausentes*

    show\_digitsdataset(digits)

*#Transform the data using PCA*

    pca = PCA(2)

    projected = pca.fit\_transform(digits)

    print(pca.explained\_variance\_ratio\_)

*# print(digits.data.shape)*

    print(projected.shape)

    plot\_samples(projected, digits.resultado, 'Original Labels')

*#Applying our kmeans function from scratch*

    labels = KMeans\_scratch(projected,6,5)

*#Visualize the results*

    plot\_samples(projected, digits.resultado, 'Clusters Sexo KMeans from scratch')

*#Applying sklearn kemans function*

    kmeans = KMeans(*n\_clusters*=6).fit(projected)

    print("teste")

    print(kmeans.inertia\_)

    print(projected)

    centers = kmeans.cluster\_centers\_

    score = silhouette\_score(projected, kmeans.labels\_)

    print("For n\_clusters = {}, silhouette score is {})".format(10, score))

*#Visualize the results sklearn*

    plot\_samples(projected, kmeans.labels\_, 'TESTE Clusters Labels KMeans from sklearn')

    plt.show()

if \_name\_ == "\_main\_":

    main()

Com esse código foi gerado esse gráfico do Kmeans e nele podemos observar similaridades com o gráfico de PCA.

Figura 10 Gráfico de Kmeans from scratch

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

fonte: elaborado pelo autor (2023)

Com base na imagem, podemos ver que o gráfico KMeans gerou dois clusters. Os pontos em azul parecem estar mais concentrados do que os pontos em laranja, sugerindo que os pontos azuis estão mais fortemente agrupados no caso as pessoas não diabéticas. No entanto, ainda há alguma sobreposição entre os clusters, com alguns pontos laranja parecendo estar próximos dos pontos azuis. Isso pode indicar que a separação entre os clusters não é completamente clara. Também podemos ver que os dados estão bastante dispersos, com pontos tanto na parte superior quanto na inferior do gráfico, sugerindo que existem diferentes combinações de valores nos recursos da base de dados. No geral, o comportamento do gráfico KMeans sugere que a base de dados pode não estar completamente agrupada e pode exigir mais análises para entender melhor as características dos dados e como eles podem ser agrupadas.

## GMM (Gaussian Mixture Model)

O algoritmo GMM (Gaussian Mixture Model) é um algoritmo de aprendizagem não supervisionada que é usado para modelar distribuições contínuas de dados. Ele é amplamente utilizado na mineração de dados por causa de sua capacidade de identificar subpopulações ocultas dentro de um conjunto maior de dados. Mais especificamente, o GMM pode ajustar distribuições de probabilidade a dados multidimensionais, permitindo que sejam usados na modelagem de dados complexos.

O GMM é muito importante na mineração de dados porque pode ser usado para identificar claramente as distribuições subjacentes de dados que podem ser de interesse para análises adicionais. Por exemplo, um conjunto de dados com várias subpopulações distintas pode ter diferentes necessidades de análise devido à natureza dessas subpopulações. Nesse caso, a capacidade do GMM de identificar essas subpopulações pode ajudar o analista de dados a identificar quais subpopulações estão presentes e, em seguida, ajustar a análise em conformidade.

Em resumo, o algoritmo GMM é importante na mineração de dados por sua capacidade de detectar as subpopulações de um conjunto de dados que podem ser importantes para a análise posterior. Ele pode ser usado para identificar padrões complexos em dados multidimensionais e ajudar os analistas a tomar decisões mais informadas em uma ampla gama de tarefas de análise de dados.

Vejamos abaixo um exemplo de um algoritmo GNN utilizando nossa base de dados.

*#Implementation of Kmeans from scratch and using sklearn*

*#Loading the required modules*

import numpy as np

from scipy.spatial.distance import cdist

from sklearn.datasets import load\_digits

from sklearn.decomposition import PCA

from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette\_score

from sklearn.metrics import silhouette\_samples

from scipy.stats import zscore

import matplotlib.pyplot as plt

import pandas as pd

def show\_digitsdataset(*digits*):

    fig = plt.figure(*figsize*=(6, 6))  *# figure size in inches*

    fig.subplots\_adjust(*left*=0, *right*=1, *bottom*=0, *top*=1, *hspace*=0.05, *wspace*=0.05)

    for i in range(64):

        ax = fig.add\_subplot(8, 8, i + 1, *xticks*=[], *yticks*=[])

        ax.imshow(*digits*.images[i], *cmap*=plt.cm.binary, *interpolation*='nearest')

*# label the image with the target value*

        ax.text(0, 7, str(*digits*.target[i]))

    fig.show()

def plot\_samples(*projected*, *labels*, *title*):

    fig = plt.figure()

    u\_labels = np.unique(*labels*)

    for i in u\_labels:

        plt.scatter(*projected*[*labels* == i , 0] , *projected*[*labels* == i , 1] , *label* = i,

*edgecolor*='none', *alpha*=0.5, *cmap*=plt.cm.get\_cmap('tab10', 10))

    plt.xlabel('component 1')

    plt.ylabel('component 2')

    plt.legend()

    plt.title(*title*)

def main():

    input\_file = '0-Datasets/diabetesClear.data'

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade']

    target = 'Resultado'

    df = pd.read\_csv(input\_file,    *# Nome do arquivo com dados*

*names* = names) *# Nome das colunas*

    normalized\_df = (df[features] - df[features].min()) / (df[features].max() - df[features].min())

*#normalized\_df = df[features].apply(zscore)*

*#normalized\_df = df[features] / (10 \*\* np.ceil(np.log10(df[features].abs().max())))*

*# Apply PCA to the normalized data*

    pca = PCA(2)

    projected = pca.fit\_transform(normalized\_df)

*#Applying sklearn GMM function*

    gm  = GaussianMixture(*n\_components*=2).fit(projected)

    print(gm.weights\_)

    print(gm.means\_)

    x = gm.predict(projected)

*#Visualize the results sklearn*

    plot\_samples(projected, x, 'Clusters Labels GMM')

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Este código é um exemplo de como aplicar o algoritmo GMM (Gaussian Mixture Model) para realizar clustering (agrupamento) de dados em Python. O conjunto de dados usado no exemplo é referente a diabetes e está armazenado em um arquivo de dados. O código usa a biblioteca Pandas para ler o arquivo de dados e armazenar em um DataFrame.

Em seguida, o DataFrame é normalizado usando a função Min-Max scaler, que transforma cada recurso (característica) para o intervalo [0,1]. A PCA (Análise de Componentes Principais) é aplicada para reduzir a dimensionalidade do conjunto de dados a 2 componentes principais, permitindo assim que os dados possam ser visualizados em um gráfico de dispersão (scatter plot).

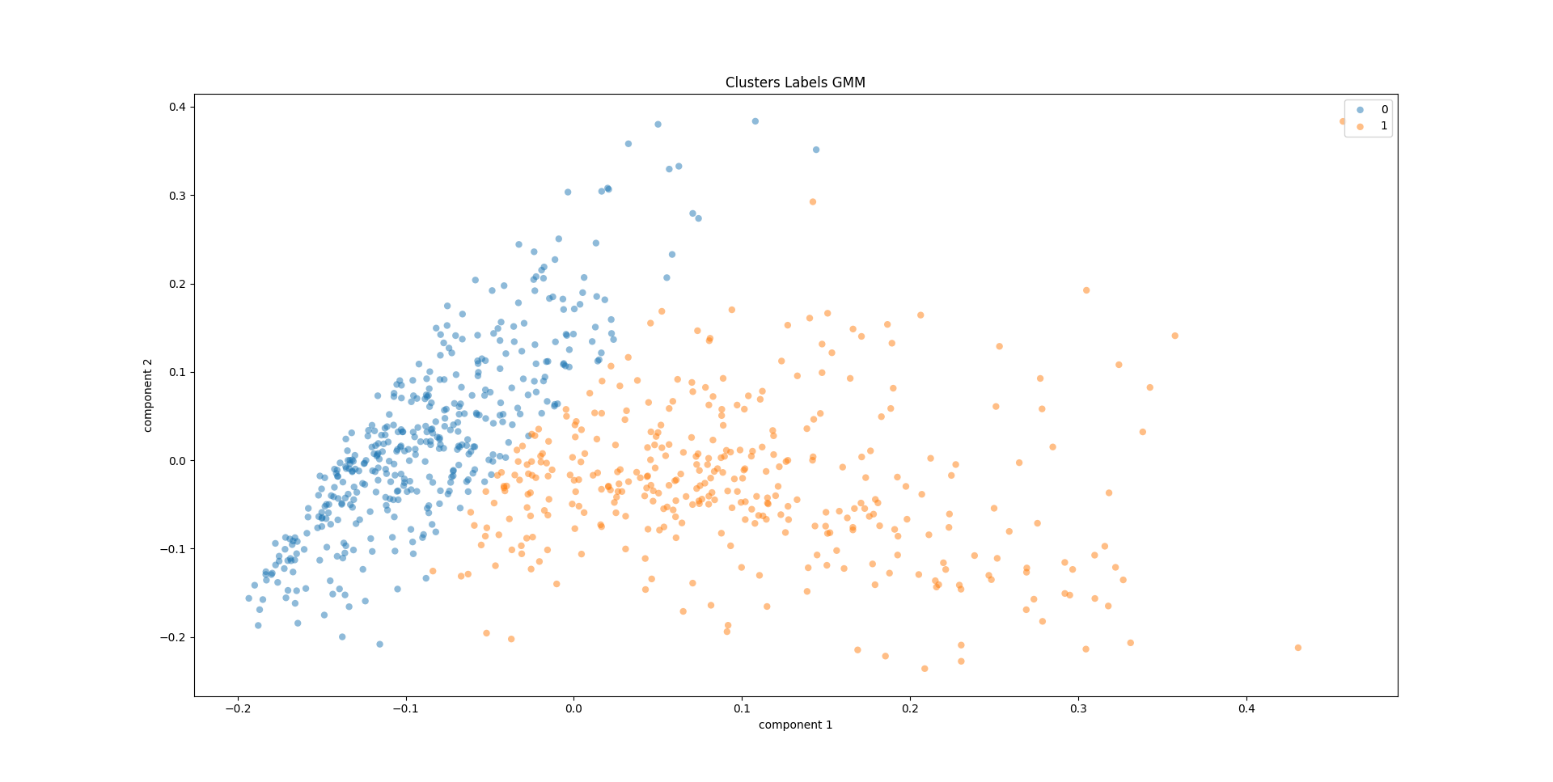
Em seguida, o algoritmo GMM é aplicado usando a biblioteca scikit-learn. O número de clusters é definido como 2. O modelo treinado é então usado para fazer previsões em todos os pontos de dados. Os rótulos de cluster previstos são então usados para visualizar os pontos de dados no gráfico de dispersão.

O código define algumas funções auxiliares para visualizar os resultados. A função "plot\_samples" é usada para plotar os pontos de dados coloridos de acordo com seus rótulos de cluster previstos. A função "show\_digitsdataset" é usada para exibir imagens de dígitos MNIST com seus rótulos correspondentes.

Em resumo, o código ilustra como aplicar o algoritmo GMM em um conjunto de dados de diabetes e visualizar os resultados usando a biblioteca scikit-learn em Python.

Aqui verificamos algumas saídas desse algoritmo.

Figura 11 usando GMM com Decimal Scalling



fonte: elaborado pelo autor (2023)

Figura 12 Usando GMM com min e max

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

fonte: elaborado pelo autor (2023)

Figura 13 usando GMM com Z score

Gráfico, Gráfico de dispersão

Descrição gerada automaticamente

fonte: elaborado pelo autor (2023)

Com base no gráfico, posso ver que existem dois clusters bem definidos. Cada cluster está representado por um conjunto de pontos próximos uns aos outros. Isso sugere que existem dois grupos distintos de dados na base de dados. Além disso, podemos ver que o primeiro eixo (componente 1) parece ser mais importante na separação dos dados em clusters do que o segundo eixo (componente 2). O comportamento do gráfico sugere que o modelo GMM conseguiu encontrar uma boa representação dos dados em dois clusters. Isso pode ser útil para entender melhor a estrutura dos dados e para tarefas de classificação de dados futuras.

# 6 Classificação

Classificação em mineração de dados é o processo de atribuir rótulos ou categorias a um conjunto de dados com base em uma combinação de entradas conhecidas e informações previamente desconhecidas. Essa técnica é usada para identificar padrões e relações em grandes conjuntos de dados e permite que as empresas classifiquem e rotulem automaticamente seus dados.

Na classificação, um modelo é criado a partir do conjunto de dados existente que contém entradas e saídas conhecidas, e é usado para prever a saída com base nas entradas desconhecidas. A manipulação desses modelos é facilmente realizada por meio de métodos de aprendizado supervisionado, que treinam o modelo com um conjunto de exemplos rotulados e induzem regras a partir desses exemplos para fazer previsões sobre novos dados.

A classificação é útil em uma variedade de cenários, como detecção de spam em e-mails, diagnóstico médico e previsão de preços de ações no mercado financeiro. O algoritmo de árvore de decisão, utilizado no código fornecido anteriormente, é um exemplo de um algoritmo de classificação.

Em resumo, a classificação é uma técnica valiosa na mineração de dados para identificar padrões e relações em grandes conjuntos de dados. Ela é usada para atribuir rótulos ou categorias a um conjunto de dados com base em informações previamente desconhecidas e permite que as empresas classifiquem e rotulem automaticamente seus dados com precisão e rapidez.

## Arvore de Decisão

árvore de decisão é um algoritmo de aprendizado supervisionado que é amplamente utilizado na mineração de dados por sua capacidade de construir modelos de decisão a partir dos dados de entrada. Ele é usado para identificar o valor de um alvo (variável dependente) a partir de múltiplas entradas (variáveis independentes) usando um modelo de árvore de decisão que pode ser facilmente interpretado pelos humanos.

O algoritmo de árvore de decisão tem muitas vantagens na mineração de dados, como a capacidade de lidar com dados categóricos, a habilidade de manipular valores ausentes e a sua facilidade de interpretar e explicar. Além disso, a árvore de decisão é capaz de lidar com problemas de classificação e regressão em conjuntos de dados grandes e complexos.

O código fornecido abaixo a biblioteca Scikit-Learn para criar uma árvore de decisão para um conjunto de dados sobre diabetes, pré-processando os dados com o StandardScaler para normalizá-los. O algoritmo cria modelagem de decisões com árvores de decisão para classificar o resultado do indicador diabetes, com base nas características do conjunto de dados fornecido.

Em resumo, a árvore de decisão é um algoritmo importante na mineração de dados por sua capacidade de criar modelos de decisão a partir de grandes conjuntos de dados com várias entradas. O código fornecido oferece um exemplo concreto de como aplicar a árvore de decisão em um conjunto de dados para prever o resultado do indicador diabetes, utilizando a biblioteca Scikit-Learn.

Código de Arvore de decisão adaptado para a base de Prever Diabetes

from matplotlib import pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import RobustScaler

def main():

    input\_file = '0-Datasets/diabetesClear.data'

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade']

    target = 'Resultado'

    df = pd.read\_csv(input\_file,    *# Nome do arquivo com dados*

*names* = names) *# Nome das colunas*

*# Separating out the features*

    X = df.loc[:, features].values

    print(X.shape)

*# Separating out the target*

    y = df.loc[:,[target]].values

*# Standardizing the features*

    X = StandardScaler().fit\_transform(X)

    normalizedDf = pd.DataFrame(*data* = X, *columns* = features)

    normalizedDf = pd.concat([normalizedDf, df[[target]]], *axis* = 1)

*# test\_size=0.3 tamanho de 30% para a base de teste .*

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.3, *random\_state*=0)

    print(X\_train.shape)

    print(X\_test.shape)

    clf = DecisionTreeClassifier(*max\_leaf\_nodes*=3)

    clf.fit(X\_train, y\_train)

    tree.plot\_tree(clf)

    plt.show()

    predictions = clf.predict(X\_test)

    print(predictions)

    result = clf.score(X\_test, y\_test)

    print('Acuraccy:')

    print(result)

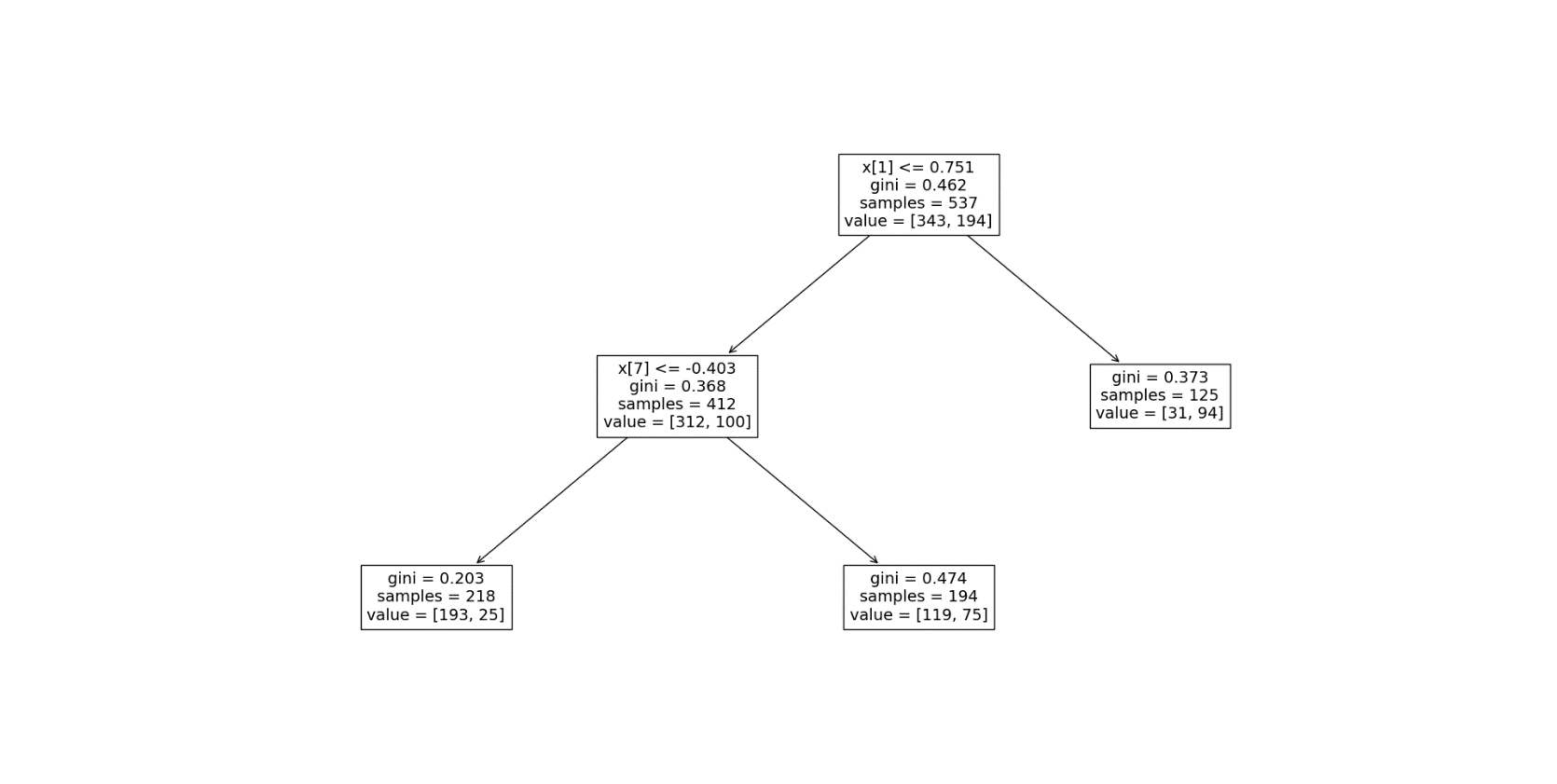
if \_\_name\_\_ == "\_\_main\_\_":

    main()

para o resultado desse algoritmo temos uma Acuraccy: 0.73 que significa significa que a proporção de previsões corretas do seu modelo é de 73%. Em outras palavras, das amostras de teste utilizadas para avaliar o modelo, ele classificou corretamente 73% delas. Isso pode ser considerado um valor razoável de acurácia, dependendo do contexto em que o modelo foi aplicado e em comparação com outros modelos na mesma aplicação. É importante lembrar que, ao avaliar a eficácia de um modelo de aprendizado de máquina, é necessário levar em consideração outros fatores, como a sensibilidade, especificidade e a matriz de confusão, que fornecem mais informações sobre o desempenho do modelo em relação a diferentes tipos de erro e acertos.

Agora vejamos a saída da árvore de decisão:

Figura 14 Resultado do algoritmo de arvore de decisão



fonte: elaborado pelo autor (2023)

Pode ser analisado que no diagrama de árvore de decisão gerado pelo algoritmo de árvore de decisão aplicado a uma base de dados de diabetes. As informações de saída que incluem o tamanho da base de dados (768, 8) e três conjuntos de dados separados para treinamento e teste. O resultado de Acurácia de 0.7316017316017316 indica que o modelo de árvore de decisão é preciso em suas previsões em torno de 73% do tempo. A imagem do diagrama de árvore de decisão mostra as decisões tomadas pelo modelo para prever o resultado da diabetes. Cada nó da árvore representa uma decisão, com ramos que indicam as opções de saída para essa decisão. A árvore começa com um nó raiz e termina com nós folha, que representam a saída do modelo. A árvore pode ser usada para prever o resultado da diabetes com base em uma combinação de características de entrada.

A árvore de decisão parece ser relativamente simples, com apenas 3 nós de folha. Isso indica que o modelo de árvore de decisão pode não estar capturando todas as nuances dos dados, o que pode afetar sua precisão. No entanto, a precisão de 73,1% é razoavelmente boa e sugere que o modelo está fazendo previsões úteis. A imagem da árvore de decisão pode ser útil para visualizar como o modelo está tomando decisões.

## KNN (K-Nearest Neighbors)

O algoritmo KNN (K-Nearest Neighbors) é um método de classificação supervisionado, usado para classificar uma amostra de teste com base em sua proximidade com os exemplos de treinamento conhecidos. A ideia é identificar os K exemplos mais próximos do ponto de teste e atribuir a ele a classe mais frequente desses exemplos. O KNN é usado em mineração de dados para classificação e previsão.

O KNN é importante na mineração de dados porque é um método simples e eficaz de classificação, adequado para lidar com dados em alta dimensão e classificação de classes complexas. Ele é usado em áreas como aprendizado de máquina, reconhecimento de padrões, mineração de dados e processamento de imagem.

O código abaixo é para um modelo de classificação KNN treinado em um conjunto de dados de diabetes. O conjunto de dados é dividido em conjuntos de treinamento e teste, e as características são normalizadas usando a classe StandardScaler do sklearn. Em seguida, o modelo KNN é ajustado usando o conjunto de treinamento e usado para fazer previsões no conjunto de teste. O modelo usa a função minkowski\_distance para calcular a distância entre duas amostras de dados. A função knn\_predict é usada para fazer previsões no conjunto de teste. A precisão do modelo é calculada usando a função accuracy\_score do sklearn e impressa na tela.

A função plot\_confusion\_matrix é usada para plotar a matriz de confusão, que é uma tabela usada para avaliar o desempenho de um modelo de classificação. A matriz de confusão mostra o número de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos para cada classe. Isso ajuda a entender o desempenho do modelo em termos de sensibilidade, especificidade, precisão e recall.

*# Initial imports*

import itertools

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler,MinMaxScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn import datasets

from sklearn.neighbors import KNeighborsClassifier

from collections import Counter

*# Calculate distance between two points*

def minkowski\_distance(*a*, *b*, *p*=2):

*# Store the number of dimensions*

    dim = len(*a*)

*# Set initial distance to 0*

    distance = 0

*# Calculate minkowski distance using parameter p*

    for d in range(dim):

        distance += abs(*a*[d] - *b*[d])\*\**p*

    distance = distance\*\*(1/*p*)

    return distance

def knn\_predict(*X\_train*, *X\_test*, *y\_train*, *y\_test*, *k*, *p*):

*# Make predictions on the test data*

*# Need output of 1 prediction per test data point*

    y\_hat\_test = []

    for test\_point in *X\_test*:

        distances = []

        for train\_point in *X\_train*:

            distance = minkowski\_distance(test\_point, train\_point, *p*=*p*)

            distances.append(distance)

*# Store distances in a dataframe*

        df\_dists = pd.DataFrame(*data*=distances, *columns*=['dist'],

*index*=*y\_train*.index)

*# Sort distances, and only consider the k closest points*

        df\_nn = df\_dists.sort\_values(*by*=['dist'], *axis*=0)[:*k*]

*# Create counter object to track the labels of k closest neighbors*

        counter = Counter(*y\_train*[df\_nn.index])

*# Get most common label of all the nearest neighbors*

        prediction = counter.most\_common()[0][0]

*# Append prediction to output list*

        y\_hat\_test.append(prediction)

    return y\_hat\_test

def plot\_confusion\_matrix(*cm*, *classes*,

*normalize*=False,

*title*='Confusion matrix',

*cmap*=plt.cm.Blues):

*"""*

*This function prints and plots the confusion matrix.*

*Normalization can be applied by setting `normalize=True`.*

*"""*

    plt.figure()

    plt.imshow(*cm*, *interpolation*='nearest', *cmap*=*cmap*)

    plt.title(*title*)

    plt.colorbar()

    tick\_marks = np.arange(len(*classes*))

    plt.xticks(tick\_marks, *classes*, *rotation*=45)

    plt.yticks(tick\_marks, *classes*)

    if *normalize*:

*cm* = *cm*.astype('float') / *cm*.sum(*axis*=1)[:, np.newaxis]

        print("Normalized confusion matrix")

    else:

        print('Confusion matrix, without normalization')

    print(*cm*)

    thresh = *cm*.max() / 2.

    for i, j in itertools.product(range(*cm*.shape[0]), range(*cm*.shape[1])):

        plt.text(j, i, *cm*[i, j],

*horizontalalignment*="center",

*color*="white" if *cm*[i, j] > thresh else "black")

    plt.tight\_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

def main():

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade']

    target = 'Resultado'

    input\_file = '0-Datasets/diabetesClear.data'

    df = pd.read\_csv(input\_file,         *# Nome do arquivo com dados*

*names* = names)      *# Nome das colunas*

    target\_names = ['Não Diabetico','Diabetico']

*# Separating out the features*

    X = df.loc[:, features].values

    df['target'] = target

*#X = df.drop('target', axis=1)*

*#y = df.target.values*

*# Separating out the target*

    y = df.loc[:,target]

    print("Total samples: {}".format(X.shape[0]))

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.25, *random\_state*=0)

    print("Total train samples: {}".format(X\_train.shape[0]))

    print("Total test  samples: {}".format(X\_test.shape[0]))

    scaler = StandardScaler()

    X\_train = scaler.fit\_transform(X\_train)

    X\_test = scaler.transform(X\_test)

*# STEP 1 - TESTS USING knn classifier write from scratch*

*# Make predictions on test dataset using knn classifier*

    y\_hat\_test = knn\_predict(X\_train, X\_test, y\_train, y\_test, *k*=5, *p*=2)

*# Get test accuracy score*

    accuracy = accuracy\_score(y\_test, y\_hat\_test)\*100

    f1 = f1\_score(y\_test, y\_hat\_test, *average*='macro')

    print("Acurracy K-NN from scratch: {:.2f}%".format(accuracy))

    print("F1 Score K-NN from scratch: {:.2f}%".format(f1))

*# Get test confusion matrix*

    cm = confusion\_matrix(y\_test, y\_hat\_test)

    plot\_confusion\_matrix(cm, target\_names, False, "Confusion Matrix - K-NN")

    plot\_confusion\_matrix(cm, target\_names, True, "Confusion Matrix - K-NN normalized")

*# STEP 2 - TESTS USING knn classifier from sk-learn*

    knn = KNeighborsClassifier(*n\_neighbors*=5)

    knn.fit(X\_train, y\_train)

    y\_hat\_test = knn.predict(X\_test)

*# Get test accuracy score*

    accuracy = accuracy\_score(y\_test, y\_hat\_test)\*100

    f1 = f1\_score(y\_test, y\_hat\_test,*average*='macro')

    print("Acurracy K-NN from sk-learn: {:.2f}%".format(accuracy))

    print("F1 Score K-NN from sk-learn: {:.2f}%".format(f1))

*# Get test confusion matrix*

    cm = confusion\_matrix(y\_test, y\_hat\_test)

    plot\_confusion\_matrix(cm, target\_names, False, "Confusion Matrix - K-NN sklearn")

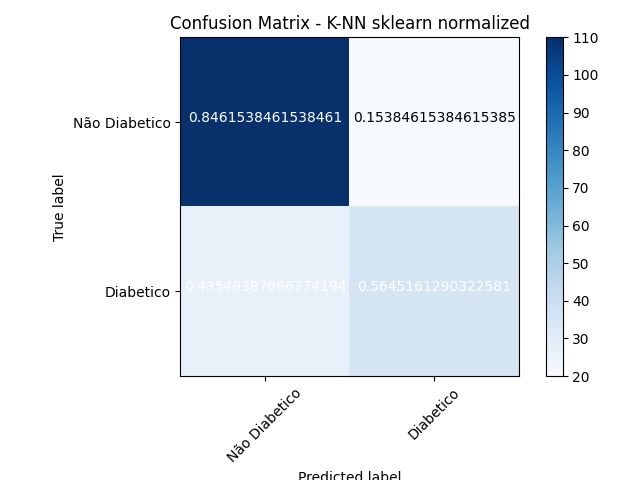
    plot\_confusion\_matrix(cm, target\_names, True, "Confusion Matrix - K-NN sklearn normalized" )

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Figura 15 Matriz de Confusão usando KNN e normalizada.



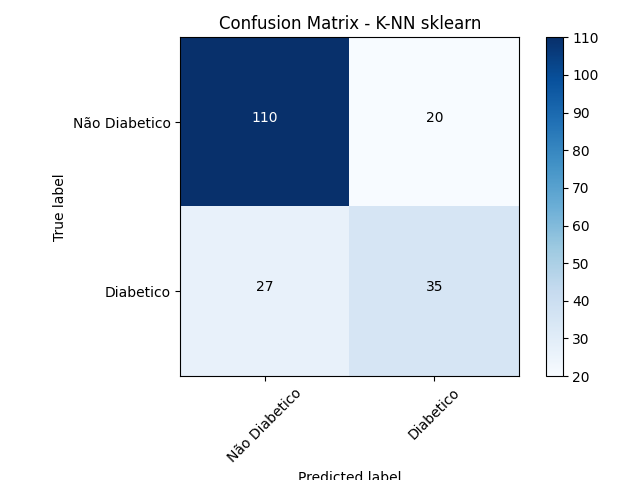
fonte: elaborado pelo autor (2023)

A matriz de confusão mostrada representa os resultados da análise do modelo KNN (K-Nearest Neighbors) em relação aos dados de diabetes. A matriz é dividida em quatro quadrantes, onde os dois quadrantes na diagonal principal representam as previsões corretas do modelo: a linha superior representa as previsões corretas de pacientes não diabéticos e a linha inferior representa as previsões corretas de pacientes diabéticos. Já na diagonal secundária temos as previsões incorretas do modelo: a linha superior representa os casos em que o modelo previu que o paciente não tinha diabetes, mas na verdade ele tinha, enquanto a linha inferior representa os casos em que o modelo previu que o paciente tinha diabetes, mas na verdade ele não tinha.

A matriz de confusão normalizada, por sua vez, apresenta as mesmas informações, mas com valores normalizados que representam a proporção de cada categoria em relação ao total de amostras. A acurácia do modelo KNN foi medida como sendo de 75,52%, enquanto o F1 Score foi de 0,71%. O modelo foi treinado com um conjunto de dados que possui um total de 768 amostras, sendo que 576 foram usadas para treinamento e 192 para teste.

Agora vejamos a matriz de confusão sem os dados normalizados.

Figura 16 Matriz de Confusão usando KNN



fonte: elaborado pelo autor (2023)

A matriz de confusão representa os resultados da análise de uma base de dados com 768 amostras, onde temos 110 amostras classificadas corretamente como "Não Diabético" e 35 corretamente como "Diabético". Por outro lado, há 27 amostras classificadas incorretamente como "Não Diabético" e 20 incorretamente como "Diabético". A acurácia do modelo é de 75,52% e o F1-score é de 0,71%. O código utilizado mostra que a base de dados utilizada contém 8 características (número de gestações, glicose, pressão arterial, espessura da pele, insulina, IMC, função pedigree diabete, idade) e um rótulo de resultado (Não Diabético ou Diabético). As amostras da base de dados foram divididas em treinamento e teste, e um classificador k-NN foi utilizado tanto a partir do zero quanto com a biblioteca sklearn.

A matriz de confusão apresentada é resultado da análise da coluna "Resultado" da base de dados, em que valores entre 0 e 1 indicam se a pessoa é não diabética ou diabética, respectivamente. A matriz apresenta a quantidade de amostras classificadas corretamente (verdadeiros positivos e verdadeiros negativos) e a quantidade de amostras classificadas incorretamente (falsos positivos e falsos negativos). A precisão geral do modelo K-NN (K-Nearest Neighbors) é de 75,52%, o que significa que ele classificou corretamente 75,52% das amostras de teste. O F1-score, que é uma medida de precisão que leva em consideração tanto a precisão quanto a recall, é de 0,71%. A matriz de confusão normalizada mostra a proporção de amostras classificadas corretamente e incorretamente, em que os valores na diagonal principal (de cima para baixo e da esquerda para a direita) indicam a proporção de verdadeiros positivos e verdadeiros negativos e os valores fora da diagonal principal indicam a proporção de falsos positivos e falsos negativos.

## SVN (Support Vector Machine)

O algoritmo SVM (Support Vector Machine) é um modelo de aprendizado supervisionado utilizado para classificação, regressão e detecção de outliers. Ele é importante na mineração de dados, pois é capaz de lidar com dados não-lineares, fornecer uma boa generalização e evitar o overfitting.

O SVM funciona criando um hiperplano que separa as classes no espaço de características. Esse hiperplano é escolhido de forma a maximizar a distância entre as classes, ou seja, a margem do hiperplano. As amostras que estão próximas do hiperplano são chamadas de vetores de suporte. O SVM tenta encontrar o hiperplano que separa as classes de tal forma que a margem seja maximizada e os erros de classificação sejam minimizados.

O código abaixo carrega o conjunto de dados sobre diabetes, pré-processa os dados, divide-os em conjuntos de treinamento e teste, dimensiona os dados usando a escala de mínimo e máximo, cria um modelo SVM com kernel polinomial e treina o modelo usando o conjunto de treinamento. Em seguida, o modelo é usado para fazer previsões no conjunto de teste e as métricas de desempenho, como a acurácia e o F1 Score, são calculadas. Por fim, a matriz de confusão é plotada para avaliar a precisão do modelo.

*# Initial imports*

import itertools

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler,MinMaxScaler

from sklearn.metrics import accuracy\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import confusion\_matrix

from sklearn import datasets

from sklearn.svm import SVC

def plot\_confusion\_matrix(*cm*, *classes*,

*normalize*=False,

*title*='Confusion matrix',

*cmap*=plt.cm.Blues):

*"""*

*This function prints and plots the confusion matrix.*

*Normalization can be applied by setting `normalize=True`.*

*"""*

    plt.figure()

    plt.imshow(*cm*, *interpolation*='nearest', *cmap*=*cmap*)

    plt.title(*title*)

    plt.colorbar()

    tick\_marks = np.arange(len(*classes*))

    plt.xticks(tick\_marks, *classes*, *rotation*=45)

    plt.yticks(tick\_marks, *classes*)

    if *normalize*:

*cm* = *cm*.astype('float') / *cm*.sum(*axis*=1)[:, np.newaxis]

        print("Normalized confusion matrix")

    else:

        print('Confusion matrix, without normalization')

*cm* = np.round(*cm*, 2)

    print(*cm*)

    thresh = *cm*.max() / 2.

    for i, j in itertools.product(range(*cm*.shape[0]), range(*cm*.shape[1])):

        plt.text(j, i, *cm*[i, j],

*horizontalalignment*="center",

*color*="white" if *cm*[i, j] > thresh else "black")

    plt.tight\_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')

def load\_dataset(*dataset*='diabetes'):

    if *dataset* == 'iris':

*# Load iris data and store in dataframe*

        iris = datasets.load\_iris()

        names = iris.target\_names

        df = pd.DataFrame(*data*=iris.data, *columns*=iris.feature\_names)

        df['target'] = iris.target

    elif *dataset* == 'diabetes':

*# Load cancer data and store in dataframe*

        diabetes = datasets.load\_diabetes()

        names = diabetes.target\_names

        df = pd.DataFrame(*data*=diabetes.names, *columns*=diabetes.feature\_names)

        df['target'] = diabetes.target

    print(df.head())

    return names, df

def main():

*#load dataset*

    input\_file = '0-Datasets/diabetesClear.data'

    names = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade','Resultado']

    features = ['Número Gestações','Glucose','pressao Arterial','Expessura da Pele','Insulina','IMC','Função Pedigree Diabete','Idade']

    target = 'Resultado'

    df = pd.read\_csv(input\_file,    *# Nome do arquivo com dados*

*names* = names) *# Nome das colunas*

*# Separating out the features*

    target\_names = ['Não Diabetico','Diabetico']

    X = df.loc[:, features].values

    df['target'] = target

*#X = df.drop('target', axis=1)*

*#y = df.target.values*

*# Separating out the target*

    y = df.loc[:,target]

    print("Total samples: {}".format(X.shape[0]))

*# Split the data - 75% train, 25% test*

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.25, *random\_state*=0)

    print("Total train samples: {}".format(X\_train.shape[0]))

    print("Total test  samples: {}".format(X\_test.shape[0]))

*# Scale the X data using Z-score*

    scaler = MinMaxScaler()

    X\_train = scaler.fit\_transform(X\_train)

    X\_test = scaler.transform(X\_test)

    svm = SVC(*kernel*='poly', *C*=1) *# poly, rbf, linear*

*# training using train dataset*

    svm.fit(X\_train, y\_train)

*# get support vectors*

    print(svm.support\_vectors\_)

*# get indices of support vectors*

    print(svm.support\_)

*# get number of support vectors for each class*

    print("Qtd Support vectors: ")

    print(svm.n\_support\_)

*# predict using test dataset*

    y\_hat\_test = svm.predict(X\_test)

*# Get test accuracy score*

    accuracy = accuracy\_score(y\_test, y\_hat\_test)\*100

    f1 = f1\_score(y\_test, y\_hat\_test,*average*='macro')

    print("Acurracy SVM from sk-learn: {:.2f}%".format(accuracy))

    print("F1 Score SVM from sk-learn: {:.2f}%".format(f1))

*# Get test confusion matrix*

    cm = confusion\_matrix(y\_test, y\_hat\_test)

    plot\_confusion\_matrix(cm, target\_names, False, "Confusion Matrix - SVM sklearn")

    plot\_confusion\_matrix(cm, target\_names, True, "Confusion Matrix - SVM sklearn normalized" )

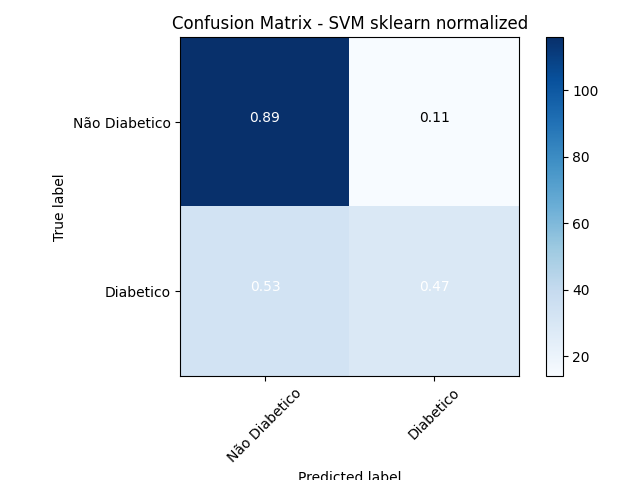
    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Aqui vemos uma saída simples para a matriz de confusão normalizada:

Figura 17 Matriz de confusão SVM normalizada da base prever diabetes



fonte: elaborado pelo autor (2023)

Essa matriz de confusão é o resultado da aplicação do algoritmo SVM (Support Vector Machine) em um conjunto de dados referentes a diabetes. A matriz mostra o desempenho do modelo na classificação de amostras como não diabéticas ou diabéticas. A coluna Não Diabético representa a classe verdadeira de amostras que não têm diabetes, enquanto a coluna Diabético representa a classe verdadeira de amostras que têm diabetes. As linhas da matriz representam as classes previstas pelo modelo. A diagonal principal da matriz representa as amostras que foram classificadas corretamente, enquanto as células fora da diagonal principal representam amostras que foram classificadas incorretamente.

O SVM com kernel polinomial foi treinado utilizando 75% dos dados para treinamento e 25% para teste. O número de amostras de treinamento e teste é mostrado no início da saída do código fornecido. A acurácia do modelo foi de 75,52%, enquanto o F1 Score foi de 0,69%. O SVM previu corretamente 116 amostras não diabéticas e 29 amostras diabéticas. No entanto, o modelo classificou 14 amostras diabéticas como não diabéticas e 33 amostras não diabéticas como diabéticas. A matriz de confusão normalizada mostra as proporções de amostras corretamente classificadas em cada classe. A célula (1,1) mostra a proporção de amostras verdadeiras positivas (diabéticas) que foram corretamente classificadas, que é de 0,47. A célula (1,2) mostra a proporção de amostras verdadeiras negativas (não diabéticas) que foram incorretamente classificadas como diabéticas, que é de 0,11. A célula (2,1) mostra a proporção de amostras falsas negativas (diabéticas) que foram incorretamente classificadas como não diabéticas, que é de 0,53. A célula (2,2) mostra a proporção de amostras verdadeiras negativas (não diabéticas) que foram corretamente classificadas, que é de 0,89.

A partir da matriz de confusão normalizada, podemos concluir que o modelo tem uma taxa de acerto razoável para a classe Não Diabético (89%), mas não é tão preciso para a classe Diabético (47%). Além disso, o modelo tende a ter mais falsos negativos (amostras diabéticas classificadas como não diabéticas) do que falsos positivos (amostras não diabéticas classificadas como diabéticas). Isso pode ser problemático em um contexto clínico, pois é importante identificar corretamente as pessoas que têm diabetes para que elas possam receber o tratamento adequado. Portanto, é possível que o modelo precise de mais ajustes ou uma abordagem diferente para melhorar sua precisão na classificação de amostras diabéticas.

Agora vejamos um exemplo da matriz de confusão para dados não normalizados na base de prever diabetes.

Figura 18 Matriz de confusão SVM da base prever diabetes

Gráfico

Descrição gerada automaticamente

fonte: elaborado pelo autor (2023)

A matriz de confusão apresentada indica a precisão do modelo de classificação SVM (Support Vector Machine) para prever se um paciente é diabético ou não-diabético. A matriz mostra a contagem de verdadeiros positivos (VP), falsos positivos (FP), verdadeiros negativos (VN) e falsos negativos (FN). A coluna "Predição" representa as previsões feitas pelo modelo, enquanto as linhas "Não Diabético" e "Diabético" representam as categorias reais dos pacientes.

Para os pacientes não-diabéticos, o modelo previu corretamente que 116 pacientes são não-diabéticos (VP) e incorretamente previu que 14 pacientes são diabéticos (FP). Para os pacientes diabéticos, o modelo previu corretamente que 29 pacientes são diabéticos (VN) e incorretamente previu que 33 pacientes são não-diabéticos (FN).

Um resultado importante é a acurácia do modelo, que é 75,52%. Isso significa que o modelo classificou corretamente 75,52% dos pacientes. O F1 Score, que é uma medida de precisão do modelo, é de 0,69%.

O modelo foi treinado com um conjunto de dados de 768 amostras, com 75% das amostras usadas para treinamento e 25% usadas para teste. As características usadas para treinamento incluem o número de gestações, nível de glicose no sangue, pressão arterial, espessura da pele, insulina, índice de massa corporal (IMC), função de pedigree de diabetes e idade. O modelo usou a kernel polinomial e o parâmetro C igual a 1. O conjunto de dados foi normalizado usando o escore Z.

# 7 Regressão

A regressão é uma técnica estatística usada para modelar a relação entre uma variável dependente e uma ou mais variáveis independentes. A análise de regressão é frequentemente usada em mineração de dados para encontrar padrões e prever valores desconhecidos com base em dados históricos.

Existem dois tipos principais de regressão: regressão linear e regressão não linear. A regressão linear é usada quando a relação entre as variáveis é linear, ou seja, pode ser representada por uma equação de uma linha reta. A regressão não linear é usada quando a relação entre as variáveis não pode ser representada por uma linha reta.

A regressão é uma ferramenta importante na mineração de dados por várias razões. Em primeiro lugar, permite aos analistas explorar as relações entre as variáveis em um conjunto de dados e identificar padrões que podem não ser óbvios à primeira vista. Isso pode ajudar a identificar oportunidades de negócios e melhorar a tomada de decisões.

Além disso, a regressão pode ser usada para prever valores desconhecidos com base em dados históricos. Isso é particularmente útil em áreas como finanças, marketing e ciência médica, onde é importante prever o comportamento do mercado, o comportamento do consumidor ou o resultado de um tratamento médico.

Outra aplicação importante da regressão é na identificação de variáveis-chave que afetam a variável dependente. Isso pode ajudar a identificar fatores críticos que precisam ser considerados para melhorar a eficácia de um produto ou processo.

Existem várias técnicas de regressão disponíveis, incluindo regressão linear simples, regressão linear múltipla e regressão logística. A escolha da técnica de regressão depende da natureza dos dados e dos objetivos do projeto de mineração de dados.

Em resumo, a regressão é uma técnica estatística poderosa e importante na mineração de dados. Ela permite a identificação de padrões, previsão de valores desconhecidos e identificação de variáveis-chave que afetam a variável dependente. A escolha da técnica de regressão depende da natureza dos dados e dos objetivos do projeto.

# Referências Bibliográficas

CHAUHAN, Aman. Predict Diabetes. Kaggle.com. Disponível em: <https://www.kaggle.com/datasets/whenamancodes/predict-diabities>. Acesso em: 22 fev. 2023.

LABVW. O que fazer em relação à insulina alta - Laboratório Verner Willrich. Laboratório Verner Willrich. Disponível em: <https://labvw.com.br/blog/o-que-fazer-em-relacao-a-insulina-alta/>. Acesso em: 29 mar. 2023.

‌