**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO - UFES**
**CURSO DE SISTEMAS DE INFORMAÇÃO**

**LAYON JOSÉ PEDROSA DOS REIS**
**WILLIAM ALVES SANTANA DA SILVA**

# TRABALHO PRÁTICO:
# Aprendizado de Máquina

**Alegre - ES**
**2023**

LAYON JOSÉ PEDROSA DOS REIS
WILLIAM ALVES SANTANA DA SILVA

# TRABALHO PRÁTICO:

## Aprendizado de Maquina

Trabalho apresentado para a Disciplina de Inteligência Artificial, pelo Curso de Sistemas de Informação da Universidade Federal do Espirito Santo (UFES), ministrada pelo prof. Jacson Rodrigues Correia da Silva

Alegre - ES
2023

**SUMÁRIO**

## 1. INTRODUÇÃO

Neste trabalho aplicaremos o conhecimento de aprendizado de máquina para criar uma Rede Neural Artificial e um algoritmo de K-NN (incluindo PCA), utilizando um dataset sobre qualidade de vinho tinto. A linguagem de programação utilizada em ambos foi python.

## 2. DATASET

O dataset escolhido é sobre vinho tinto variante de um vinho português chamado “Vinho Verde”. Este dataset pode ser visto com uma tarefa de classificações ou regressão. As classes estão ordenadas e não balenceados ( tem muito mais vinhos normais do que vinhos ruins ou exelentes) .

Tabela 1: Dados sobre o Dataset

| | | | | | |
|---|---|---|---|---|---|
| **Data Set Characteristics:** | Multivariate | **Number of Instances:** | 1599 | **Area:** | Business |
| **Attribute Characteristics:** | Real | **Number of Attributes:** | 12 | **Date Donated** | 2009-10-07 |
| **Associated Tasks:** | Classification, Regression | **Missing Values?** | N/A | **Number of Web Hits:** | 2072975 |

Fonte: Paulo Cortez, University of Minho, Guimarães, Portugal, http://www3.dsi.uminho.pt/pcortez A. Cerdeira, F. Almeida, T. Matos and J. Reis, Viticulture Commission of the Vinho Verde Region(CVRVV), Porto, Portugal @2009.

### 2.1 Informações sobre os atributos:

#### 2.1.1 Acidez fixa

A acidez fixa corresponde ao conjunto de ácidos orgânicos de baixa volatilidade como os ácidos málico, lático, tartárico ou cítrico e é inerente às características da amostra.

### 2.1.2 Acidez volátil:

A acidez volátil é um importante parâmetro sensorial, com níveis mais altos indicando deterioração do vinho.

### 2.1.3 Ácido cítrico

O ácido cítrico é frequentemente adicionado aos vinhos para aumentar a acidez, complementar um sabor específico ou evitar turvações férricas. Pode ser adicionado a vinhos acabados para aumentar a acidez e dar um sabor fresco.

### 2.1.4 Açúcar residual

O Açúcar Residual é proveniente dos açúcares naturais da uva que sobram no vinho após o término da fermentação alcoólica.

### 2.1.5 Cloreto

Eles são os principais contribuintes para o sabor salgado de um vinho.

### 2.1.6 Dióxido de enxofre livre

O dióxido de enxofre (SO2) preserva o vinho, evitando a oxidação e escurecimento.

### 2.1.7 Dióxido de enxofre total

O Dióxido de Enxofre Total (TSO2) é a porção de SO2 que está livre no vinho mais a porção que está ligada a outras substâncias químicas no vinho, como aldeídos, pigmentos ou açúcares.

### 2.1.8 Densidade

Mantendo o nível de álcool constante, a densidade tem pouco efeito na qualidade dos vinhos, pois outras coisas podem contribuir na densidade.

### 2.1.9 pH

Os produtores de vinho usam o pH como uma forma de medir a maturação em relação à acidez. Os vinhos com pH baixo terão sabor ácido e fresco, enquanto os vinhos com pH mais alto são mais suscetíveis ao crescimento bacteriano.

### 2.1.10 Sulfatos

Pensa-se que a presença de outro tipo de sulfato ajuda a livrar o vinho de uma grande variedade de bactérias (boas e más). Isso também parece diminuir a qualidade do vinho porque embota o processo de fermentação do vinho.

### 2.1.11 Álcool (Baseado em dados sensoriais)

O teor alcoólico afeta o corpo do vinho, pois o álcool é mais viscoso que a água. Um vinho com maior teor alcoólico terá um corpo mais encorpado e rico, enquanto um vinho com baixo teor alcoólico terá um sabor mais leve e delicado no paladar.

### 2.1.12 Qualidade

A qualidade do vinho depende principalmente do processo de vinificação e da origem geográfica das uvas, mas também depende muito da composição varietal da uva.

## 2.2 Trabalhando com o dataset

### 2.2.1 Importando biblioteca

Código 1:

```
import pandas as pd
import csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

### 2.2.2 Carregando data set e removendo dados irrelevantes

Código 2:

```
# Criando data set e removendo colunas com baixa correlaçãoe duplicatas

dataset = pd.read_csv('wine-quality.csv', delimiter=';')

dataset.drop(columns=["fixed acidity", "residual sugar", "free sulfur dioxide", "pH"])

dataset.drop_duplicates()

classification = []

for x, y in enumerate(dataset["quality"]):
  if y > 5:
    classification.append(1)
  else:
    classification.append(0)

dataset["class"] = classification
```

Como mostrado no código 2, nós importamos o dataset para realizarmos o tratamento dos dados. Primeiramente removemos algumas colunas de atributos com baixa correlação, e usamos a função drop_duplicates() para remover dados duplicados, poderíamos também ter tirado dados com valor nulo, porém como nosso dataset não possuía dados com valores nulos então não houve necessidade.

O próximo passo foi criar uma classificação binária entre bom ou ruim (1 ou 0 respectivamente) baseado na qualidade do vinho, se a qualidade fosse maior que cinco a classificação era boa, caso contrário seria classificado como ruim.

### 2.2.4 Standartization

Código 3:

```
# Standartization

std_data = scaler.fit_transform(dataset)

std_df = pd.DataFrame(std_data)

for x, name in enumerate(dataset.columns):
  std_df.rename(columns={x: name}, inplace=True)

x = std_df.drop(columns=["class", "quality"])
y = dataset["class"]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

No código 3 foi feita a padronização dos dados e a divisão dos datasets de treino e teste.

### 2.2.5 Salvando como .cvs

Código 4:

```
# Salvando como .csv

with open("x_train_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(x_train.columns.values)
    writer.writerows(x_train.values.tolist())
    file.close()
with open("x_test_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(x_test.columns.values)
    writer.writerows(x_test.values.tolist())
    file.close()
with open("y_train_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["class"])
    for x in y_train:
      writer.writerow([x])
    file.close()
with open("y_test_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["class"])
    for x in y_test:
      writer.writerow([x])
    file.close()
```

## 2.3 Análise exploratória de dados

### 2.3.1 Tabela exemplo do dataset

Código 5:

```
train_data.head(10)
```

Tabela 2:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| 1 | 7.8 | 0.88 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 | 5 |
| 2 | 7.8 | 0.76 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.9970 | 3.26 | 0.65 | 9.8 | 5 |
| 3 | 11.2 | 0.28 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.9980 | 3.16 | 0.58 | 9.8 | 6 |
| 4 | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| 5 | 7.4 | 0.66 | 0.00 | 1.8 | 0.075 | 13.0 | 40.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 |
| 6 | 7.9 | 0.60 | 0.06 | 1.6 | 0.069 | 15.0 | 59.0 | 0.9964 | 3.30 | 0.46 | 9.4 | 5 |
| 7 | 7.3 | 0.65 | 0.00 | 1.2 | 0.065 | 15.0 | 21.0 | 0.9946 | 3.39 | 0.47 | 10.0 | 7 |
| 8 | 7.8 | 0.58 | 0.02 | 2.0 | 0.073 | 9.0 | 18.0 | 0.9968 | 3.36 | 0.57 | 9.5 | 7 |
| 9 | 7.5 | 0.50 | 0.36 | 6.1 | 0.071 | 17.0 | 102.0 | 0.9978 | 3.35 | 0.80 | 10.5 | 5 |

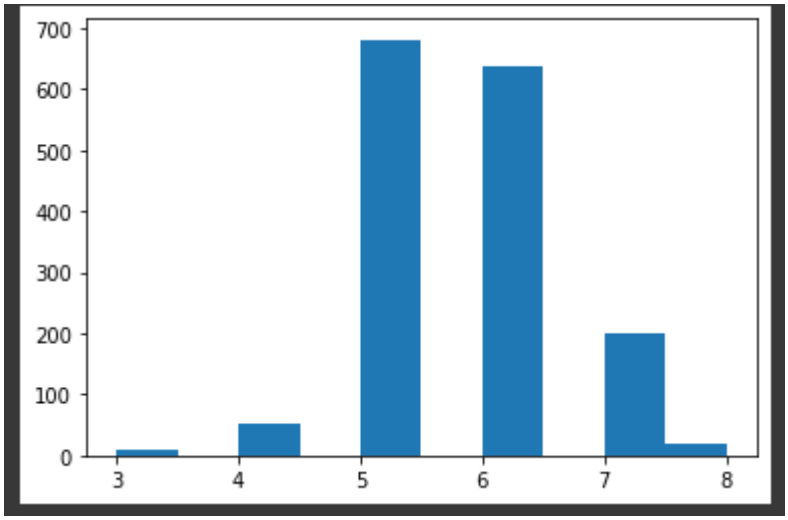
Fonte: Output do código 6.

### 2.3.2 Gráfico de distribuição dos valores de qualidade

Código 6:

```
fig=plt.figure()
ax=fig.add_subplot(1,1,1)
ax.hist(train_data["quality"],bins=10)
plt.show()
```

Gráfico 1:



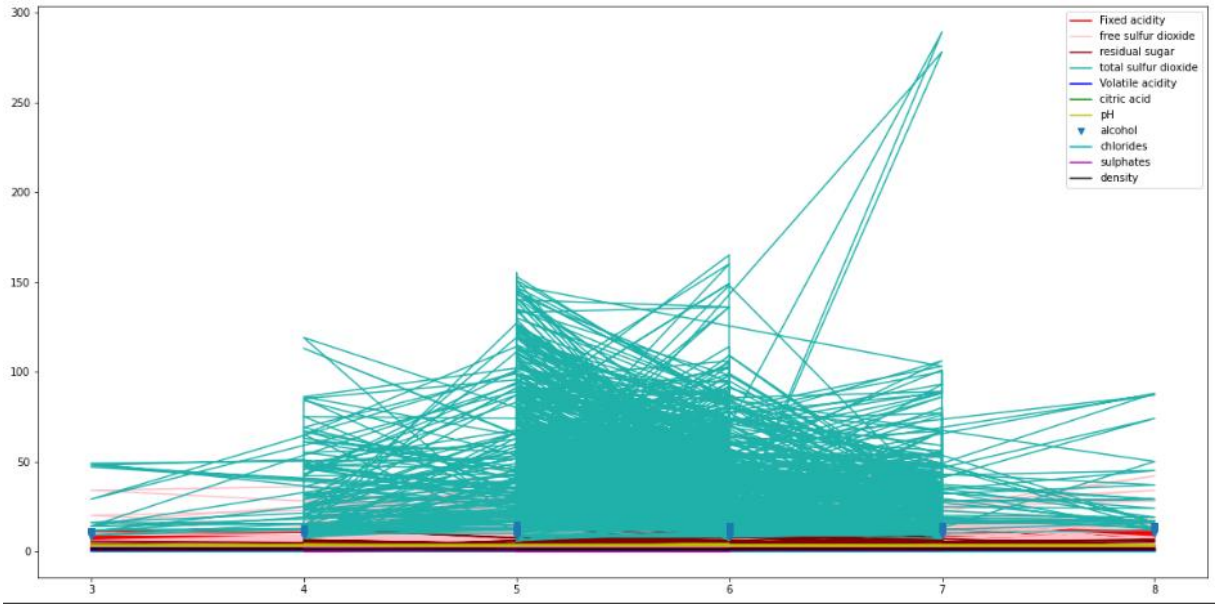

Fonte: Output do código 7.

### 2.3.3 Exploração completa de dados

Código 7:

```
f = plt.figure()
f.set_figwidth(20)
f.set_figheight(10)
x=train_data['quality']
plt.plot(x,train_data['fixed acidity'],'r',label='Fixed acidity')
plt.plot(x,train_data['free sulfur dioxide'],'pink',label='free sulfur dioxide')
plt.plot(x,train_data['residual sugar'],'maroon',label='residual sugar')
plt.plot(x,train_data['total sulfur dioxide'],'lightseagreen',label='total sulfur dioxide')
plt.plot(x,train_data['volatile acidity'],'b',label='Volatile acidity')
plt.plot(x,train_data['citric acid'],'g',label='citric acid')
plt.plot(x,train_data['pH'],'y',label='pH')
plt.plot(x,train_data['alcohol'],'v',label='alcohol')
plt.plot(x,train_data['chlorides'],'c',label='chlorides')
plt.plot(x,train_data['sulphates'],'m',label='sulphates')
plt.plot(x,train_data['density'],'k',label='density')
plt.legend(loc=0)
plt.figure()
plt.show()
```

Gráfico 2:




Fonte: Output do código 8.

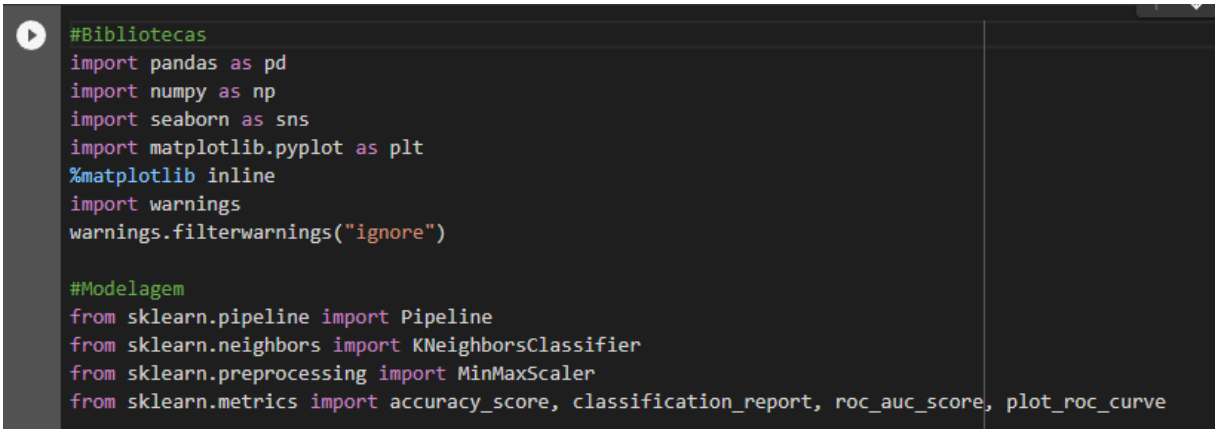
Neste gráfico podemos ver uma exploração completa dos dados. A maioria dos atributos não apresenta muita variância, porém o dióxido de enxofre total apresenta grande variedade de valores.

## 3. K-NN

KNN (K - Nearest Neighbors) é um dos muitos algoritmos (aprendizagem supervisionada) usados no campo de mineração de dados e aprendizado de máquina, é um classificador no qual o aprendizado é baseado no relacionamento de um dado (vetor) com outro. O treinamento é formado por vetores de n dimensões.

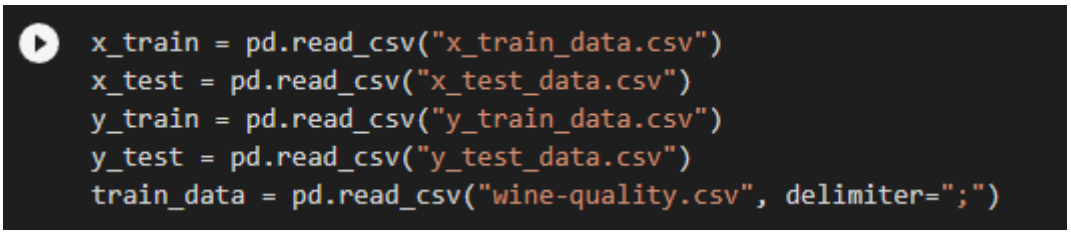
### 3.1 Código

#### 3.1.1 Importando Bibliotecas



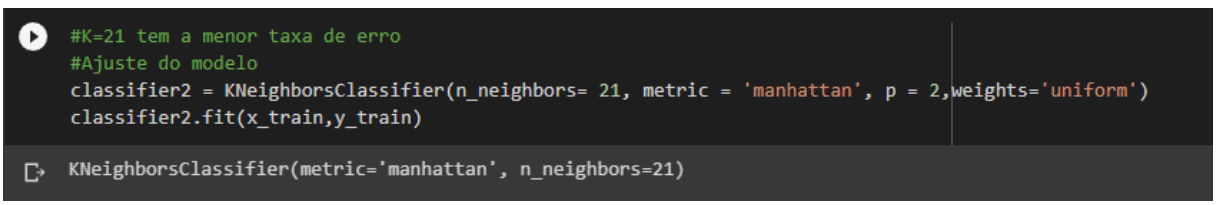
```
#Bibliotecas
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

#Modelagem
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, plot_roc_curve
```

#### 3.1.2 Importando o dataset



```
x_train = pd.read_csv("x_train_data.csv")
x_test = pd.read_csv("x_test_data.csv")
y_train = pd.read_csv("y_train_data.csv")
y_test = pd.read_csv("y_test_data.csv")
train_data = pd.read_csv("wine-quality.csv", delimiter=";")
```

#### 3.1.3 Ajuste do modelo



```
#K=21 tem a menor taxa de erro
#Ajuste do modelo
classifier2 = KNeighborsClassifier(n_neighbors= 21, metric = 'manhattan', p = 2,weights='uniform')
classifier2.fit(x_train,y_train)
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=21)
```

### 3.1.4 Prevendo a saída dos dados de entrada

```
#Prevendo a saída dos dados de entrada (x_train) e (y_train)
y_pred1 = classifier2.predict(x_train)
y_pred2 = classifier2.predict(x_test)
```

### 3.1.5 Acurácia do treino e do teste

```
from sklearn.metrics import accuracy_score
print("Accuracy score of train data set:",accuracy_score(y_train, y_pred1))
print("Accuracy score of test data set:",accuracy_score(y_test, y_pred2))
```

```
Accuracy score of train data set: 0.7444794952681388
Accuracy score of test data set: 0.7450980392156863
```

### 3.1.6 Tabela dos valores de qualidade

```
train_data['quality'].value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

### 3.1.7 Divisão dos valores de qualidade

```
#Se o valor de qualidade for menor ou igual a 5, então ele vai ficar na classe 0
#Se o valor de qualidade for maior que 5, então ele vai ficar na classe 1
train_data['quality'] = np.where(train_data['quality'] > 5, 1, 0)
train_data['quality'].value_counts()
```

```
1    855
0    744
Name: quality, dtype: int64
```

### 3.1.8 Atribuindo dataframe à lista de valores de matriz

```
#Atribuindo dataframe à lista de valores de matriz
X = train_data.drop(['quality'], axis = 1).values
y = train_data['quality'].values
```

### 3.1.9 Resultado final

```
k = range(1,50,2)
testing_accuracy = []
training_accuracy = []
score = 0
#Fitting the model
for i in k:
    knn = KNeighborsClassifier(n_neighbors = i)
    pipe_knn = Pipeline([('scale', MinMaxScaler()), ('knn', knn)])
    pipe_knn.fit(x_train, y_train)

    y_pred_train = pipe_knn.predict(x_train)
    training_accuracy.append(accuracy_score(y_train, y_pred_train))

    y_pred_test = pipe_knn.predict(x_test)
    acc_score = accuracy_score(y_test,y_pred_test)
    testing_accuracy.append(acc_score)

    if score < acc_score:
        score = acc_score
        best_k = i

print('Best Accuracy Score', score, 'Best K-Score', best_k)
```

## 4. REDES NEURAIS

As redes neurais são uma abordagem da inteligência artificial que ensina os computadores a processar dados de maneiras inspiradas no cérebro humano. É um processo de aprendizado de máquina chamado aprendizado profundo que usa nós interconectados, ou neurônios, em uma estrutura em camadas, semelhante ao cérebro humano. As redes neurais criam um sistema adaptativo que os computadores podem usar para aprender com seus erros e melhorar continuamente.

As redes neurais artificiais tentam resolver problemas complexos, como resumir documentos ou reconhecer rostos com alta precisão.

O nosso modelo de rede neural foi feito usando o keras do tensorflow. Primeiro importei os dados que seriam usados para o treino e teste, divididos entre, x e y, sendo x os inputs e y a resposta esperada, atribuí um valor ao learning rate e depois carreguei um modelo sequencial em uma variável chamada model, em seguida adicionei as camadas da minha rede neural. As camadas de entrada foram criadas automaticamente, com base no número de atributos do nosso dataset de treino, em seguida, criei duas camadas ocultas e uma última camada de saída contendo apenas um neurônio. No nosso modelo foi usado apenas a função sigmoide como função de ativação, e a resposta de saída da nossa rede foi definida como 0 ou 1, sendo 0 para baixa qualidade do vinho e 1 para boa qualidade, depois de definir os hiperparametros compilei o modelo usando uma função de “gradient descent” e para função de perda usei “binary cross entropy” e como métrica a acurácia. Ápos todas essas etapas carreguei meu modelo para iterar por 1000 épocas e medi seu desempenho com uma função de avaliação do próprio keras, após uma bateria de treinos e testes minha rede neural estava apresentando overfitting então decidi simplificar a sua estrutura diminuindo o número de neurônios nas camadas ocultas, conseguindo corrigir o overfitting e mantendo uma acurácia de mais de 70%, chegando aos seguintes valores de hiperparâmetros: 4 neurônios na primeira camada oculta e 6 na segunda camada oculta e 1 neurônio na camada de saída, com um tamanho de treino de 250 épocas e 32 bathces padrão da própria biblioteca e learning rate de 0.01.

## 4.1 Código

### 4.1.1 Importar Bibliotecas

```
import pandas as pd
import numpy as np
import csv
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

scaler = StandardScaler()
```

### 4.1.2 Tratamento dos dados e divisão em arquivos csv

```
dataset = pd.read_csv('wine-quality.csv', delimiter=';')

dataset.drop_duplicates(inplace=True)

classification = []

for x, y in enumerate(dataset["quality"]):
  if y > 5:
    classification.append(1)
  else:
    classification.append(0)

dataset["class"] = classification

std_data = scaler.fit_transform(dataset)

std_df = pd.DataFrame(std_data)

for x, name in enumerate(dataset.columns):
  std_df.rename(columns={x: name}, inplace=True)
```

```
std_df.drop(columns=["fixed acidity", "residual sugar", "free sulfur dioxide",
                     "pH"])
x = std_df.drop(columns=["class", "quality"])
y = dataset["class"]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
with open("x_train_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(x_train.columns.values)
    writer.writerows(x_train.values.tolist())
    file.close()
with open("x_test_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(x_test.columns.values)
    writer.writerows(x_test.values.tolist())
    file.close()
with open("y_train_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["class"])
    for x in y_train:
      writer.writerow([x])
    file.close()
with open("y_test_data.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerow(["class"])
    for x in y_test:
      writer.writerow([x])
    file.close()
```

4.1.3 Importação dos datasets

```
x_train = pd.read_csv("x_train_data.csv")
x_test = pd.read_csv("x_test_data.csv")
y_train = pd.read_csv("y_train_data.csv")
y_test = pd.read_csv("y_test_data.csv")
```

4.1.4 Definindo Learning rate por meio da função de otimização

```
optimizer = tf.keras.optimizers.Adam(lr=0.01)
```

4.1.5 Criando modelo sequencial, adicionando camadas e compilando

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(3,input_shape=x_train.shape[1:],
                                activation="sigmoid"))
model.add(tf.keras.layers.Dense(4, activation="sigmoid"))
model.add(tf.keras.layers.Dense(1, activation="sigmoid"))

model.compile(optimizer=optimizer, loss='binary_crossentropy',
              metrics=['accuracy'])
```

4.1.6 Testando o modelo e avaliando em seguida

```
model.fit(x_train, y_train, epochs=250)
print("\nEvaluate:")
model.evaluate(x_test, y_test)
```

4.1.7 Saída da função de avaliação

```
Evaluate:
13/13 [==============================] - 0s 3ms/step - loss: 0.5234 - accuracy: 0.7279
[0.5233911275863647, 0.7279411554336548]
```

# 5. DISCUSSÃO

## 5.1 Discussão dos resultados

No KNN, o k-score máximo foi de 41, já a acurácia foi de 75,49%, com desvio padrão de 0.

```
Best Accuracy Score 0.7549019607843137 Best K-Score 41
```

Já na RNA:

Teste 1:

```
Epoch 250/250
30/30 [==============================] - 0s 2ms/step - loss: 0.4632 - accuracy: 0.7655

Evaluate:
13/13 [==============================] - 0s 2ms/step - loss: 0.5431 - accuracy: 0.7500
[0.5431491732597351, 0.75]
```

Teste 2:

```
Epoch 250/250
30/30 [==============================] - 0s 2ms/step - loss: 0.4581 - accuracy: 0.7592

Evaluate:
13/13 [==============================] - 0s 2ms/step - loss: 0.5599 - accuracy: 0.7377
[0.5599343776702881, 0.7377451062202454]
```

Teste 3:

```
Epoch 250/250
30/30 [==============================] - 0s 2ms/step - loss: 0.4525 - accuracy: 0.7571

Evaluate:
13/13 [==============================] - 0s 2ms/step - loss: 0.5741 - accuracy: 0.7279
[0.5741389989852905, 0.7279411554336548]
```

Desvio padrão dos testes: 0.01

Acurácia média da RNA: 0.738

Em ambos os modelos não houve overfitting nem underfitting, os valores de acurácia dos testes e treinos foram bem parecidos.

### 5.1.1 Comparando os resultados

Os modelos tiveram desempenhos bastante parecidos, o KNN foi um pouco mais preciso do que a rede neural, com a diferença de 1,6% de precisão.

## 5.2 Discussão das dificuldades do trabalho

### 5.2.1 Otimização de hiperparâmetros:

Ajustar os hiperparâmetros do modelo foi a tarefa mais demorada e difícil do trabalho, especialmente quando se tratou de redes neurais.

### 5.2.2 Conjunto de dados de treinamento

Foi difícil de tratar e organizar o dataset para ajustar nos dois modelos, pois tivemos que criar arquivos de treino e de teste separados para usar em ambos os sistemas de aprendizado.

### 5.2.3 Interpretação dos resultados

Por fim, foi importante interpretar corretamente os resultados do modelo, especialmente quando se trata de redes neurais, já que sua funcionamento interno é complexo e pode ser difícil de compreender, o que tornou difícil ajustamentos dos hiperparâmetros para corrigir o overfitting.

## 6. CONCLUSÃO

Neste trabalho, foi realizada uma comparação entre o algoritmo de aprendizado de máquina KNN (K-Nearest Neighbors) e as Redes Neurais Artificiais para classificação de dados. Os resultados obtidos indicam que, em geral, o KNN apresentou melhor desempenho em comparação a RNA. No entanto, é importante destacar que a escolha do algoritmo ideal depende do conjunto de dados em questão e, portanto, é necessário avaliar as características específicas do problema antes de tomar a decisão sobre qual técnica utilizar. Além disso, foi possível perceber que a combinação dos dois algoritmos pode ser uma alternativa interessante para soluções de aprendizado de máquina mais eficientes.

Neste trabalho aprendemos muito sobre os sistemas de aprendizado KNN e Redes Neurais Artificiais, principalmente na parte de colocar em pratica os conceitos que aprendemos na sala de aula com o nosso lindo e incrivel professor Jacson Rodrigues Correia da Silva.

# REFERÊNCIAS

Paulo Cortez, University of Minho, Guimarães, Portugal, http://www3.dsi.uminho.pt/pcortez A. Cerdeira, F. Almeida, T. Matos and J. Reis, Viticulture Commission of the Vinho Verde Region(CVRVV), Porto, Portugal @2009.

Ishan Kotian, Student at RAIT, Mumbai, Maharashtra, Índia. https://www.kaggle.com/code/lykin22/red-wine-quality-analysis-knn-88-acc . Ano de 2021.

Slides de aula do Professor Jacson Rodrigues Correia da Silva, https://drive.google.com/drive/folders/1acVXms6eeTgBEzg4op7sZHeGv7HdUDhL?usp=sharing

Vídeos do Khanrad: https://www.youtube.com/watch?v=z1PGJ9quPV