UMA INTRODUÇÃO A *MACHINE LEARNING* UTILIZANDO LINGUAGEM PROGRAMAÇÃO PYTHON

Macedo da Silva, L.[1], de Almeida Silva, V.[2]

1Escola de Ciências Exatas e da Computação
Pontificia Univervidade Catolica De Goiás
Goiânia – GOIÁS – Brasil
2Escola de Ciências Exatas e da Computação
Pontificia Univervidade Catolica De Goiás
Goiânia – GOIÁS - Brasil

RESUMO: No presente trabalho, foi introduzida a aprendizagem de máquina utilizando a linguagem de programação Python, com o objetivo de tornar as técnicas aqui discutidas mais acessíveis a comunidade cientifica em geral. O banco de dados utilizado para o desenvolvimento foi o *dataset* de íris presente na biblioteca *Seaborn* da linguagem de programação Python. O procedimento utilizado foi o seguinte: 1) Análise dos dados: Verifica a integridade dos mesmos; 2) Escolha do modelo: O modelo escolhido foi o *k Nearest Neighbours* (*k* vizinhos mais próximos), devido o mesmo ser simples e prático para uma introdução; 3) Divisão dos dados: Os dados foram divididos em dois subconjuntos, o subconjunto de teste e o subconjunto de treino; 4) Treino do modelo: O modelo foi treinado com os dados de treino; 5) Avaliação do modelo: O modelo desenvolvido depois de treinado, foi avaliado, buscando verificar a performance do mesmo. Inicialmente são apresentados conceitos importantes sobre *machine learning* (aprendizagem de máquina), após é apresentado e discutido o fluxograma de atividades desenvolvidas durante a modelação. O último passo consiste no desenvolvimento do modelo em si, são apresentados alguns trechos do código desenvolvido.

PALAVRAS-CHAVE: Aprendizagem de máquina; modelo; Python; dataset.

ABSTRACT: In the present work, machine learning was introduced using the Python programming language, with the aim of making the techniques discussed here more accessible to the scientific community in general. The database used for development was the rainbow dataset present in the Seaborn library of the Python programming language. The procedure used was as follows: 1) Data analysis: Check the integrity of the data; 2) Choice of model: The model chosen was *k* Nearest Neighbors, because it is simple and practical for an introduction; 3) Data

division: The data were divided into two subsets, the test subset and the training subset; 4) Model training: The model was trained with the training data; 5) Evaluation of the model: The model developed after training, was evaluated, seeking to verify the performance of the same. Initially important concepts about machine learning are presented, after which the flowchart of activities developed during the modeling is presented and discussed. The last step consists of the development of the model itself, some sections of the developed code are presented.

KEYWORDS: Machine learning; model; Python; dataset.

1 Introdução

A Aprendizagem de máquina, também conhecida como *machine learning* ou *statical learning*, trata-se de uma das áreas da inteligência artificial que tem como objetivo, estudar e desenvolver métodos que buscam fazer com que o computador possa aprender através de dados que lhe são oferecidos. Portanto, a *machine learning* trata-se de um método de análise de dados para criação de modelos para predizer ou estimar uma saída baseada nos dados de entrada [1].

O aprendizado que o computador adquire, pode ser obtido a partir de técnicas de aprendizagem estudadas na área. Entre tais técnicas destacam-se a aprendizagem supervisionada e a aprendizagem não supervisionada, na primeira sabe-se as respostas esperadas, já na segunda não. A aplicação do conhecimento adquirido pelo computador é utilizada buscando auxiliar os seres humanos nas mais diversas tarefas, desde o reconhecimento facial [2] até mesmo no diagnóstico de câncer [3].

As grandes aplicações que as técnicas de *machine learning* permitem, exigem em sua grande parte o conhecimento dos métodos por parte do cientista. Tais métodos em sua grande maioria são complexos e seu implementação em cada aplicação pode ser demorada demandando um grande montante de tempo que poderia ser utilizado com outra finalidade. A linguagem de programação Python, é atualmente, uma das principais ferramentas no quesito de aplicações de técnica de *machine learning*, pois conta com os métodos implementados em suas bibliotecas, destacando-se a biblioteca *scikit learning*, utilizada neste trabalho.

Além do mais, a linguagem conta com várias funcionalidades voltadas para o desenvolvimento e aplicação de conhecimentos inerentes a aprendizagem de máquina. Somada a este fato, a mesma é a mais utilizada no mundo por cientistas de *machine learning* e *data Science*. Sendo, portanto, um conhecimento de suma importância, dada as aplicações que a

aprendizagem de máquina possui, podendo seu uso, tornar o desenvolvimento de novas tecnologias mais simples e rápido.

O principal objetivo do presente trabalho é o de desenvolver um classificador que seja capaz de determinar a qual espécie uma flor de íris pertence, baseado em quatro características: comprimento e largura das sépalas e comprimento e largura das pétalas. Para atingir o objetivo será utilizada o classificador *k Nearest Neighbors*, a ferramenta utilizada será a linguagem de programação Python.

2 Conceitos sobre *Machine Learning*

A *machine learning* é dividida em dois tipos de aprendizado o supervisionado quando o ser humano faz o papel de supervisor e verifica se as respostas obtidas condizem com os resultados esperados e o não supervisionado, quando não existem as respostas, o modelo então tenta encontrar algum padrão nos dados. No presente trabalho foi utilizada a aprendizagem supervisionada.

2.1 Aprendizagem supervisionada

A aprendizagem supervisionada ocorre quando o computador é treinado com exemplos e as respostas para aquele exemplo [4]. Nesse cenário o ser humano faz o papel de supervisor e verifica se os resultados estão corretos, podendo então, modificar algum parâmetro do modelo, fazendo assim com que se obtenha o resultado esperado. Sua aplicação se dá em problemas de regressão e classificação.

Sendo assim, o objetivo na aprendizagem supervisionada é encontrar um modelo que responda corretamente baseado nos dados em que foi treinado. Como sabe-se a resposta esperada, o treino é mais simples do ponto de vista humano e o mesmo fica a par de alterar algum parâmetro do modelo para alcançar uma melhor performance.

2.2 Aprendizagem não supervisionada

A aprendizagem não supervisionada trata-se do sub-ramo da *machine learning*, onde o computador tende a resolver os problemas sozinho, visto que os dados não possuem uma resposta concreta [5]. Como os dados não possuem rótulos, os mesmos são organizados em grupos, denominados na literatura de *clusters*, de forma que permitam inferir algo sobre os mesmos [6].

Sendo assim, tal métrica de ensino se baseia na construção de modelos que possuem entrada, mas nenhuma saída de supervisão. Com isso pode se aprender e estruturar relacionamentos a partir dos dados [1]. Tendo vasta aplicação em problemas de agrupamento, quando deseja-se inferir algo sobre os dados.

2.3 Overfitting

O *overfitting* também conhecido como sobre-ajuste, é um fenômeno que ocorre quando um método gera um erro pequeno para os dados de treino, já para os dados de teste gera um erro grande [6]. Em outras palavras tal problema ocorre quando o modelo se adequa muito aos dados de treino e tende a predizer incorretamente os dados de teste.

Para evitar tal fenômeno foi utilizada a separação dos dados em dois subconjuntos o subconjunto de treino e o subconjunto de testes, onde 30% dos dados foram utilizados para teste e os 70% restantes para treino.

3 Conceitos matemáticos

A matemática é um dos pilares da ciência no geral. A mesma fornece técnicas e métodos necessários para avaliar, modelar, testar, entre outros processos presentes em qualquer projeto de cunho cientifico [7]. De fato, todos os métodos de aprendizado de máquina estudados, são em sua base aplicação dos mais variados conceitos matemáticos.

A mesma foi utilizada no corrente trabalho, principalmente na análise dos resultados obtidos durante a execução de algum método.

3.1 Problema de classificação

O modelo desenvolvido busca resolver um problema de classificação. Os problemas de classificação, são categorizados como problemas que envolvem variáveis qualitativas [1]. Uma variável por sua vez, é determinada como qualitativa quando são utilizadas para representar uma classificação dos indivíduos [8]. São baseadas em qualidades e não podem ser mensuradas numericamente. Alguns exemplos de variáveis qualitativas são: A classe social a qual uma pessoa pertence e se uma pessoa possui ou não uma doença.

Portanto, o objetivo será encontrar um modelo que melhor classifique a qual espécie uma íris pertence, os rótulos serão as espécies, enquanto as demais variáveis serão utilizadas para a classificação.

3.2 Métricas de avaliação

Para avaliar o desempenho do modelo foram utilizadas duas métricas a matriz de confusão e a precisão, ambas implementadas já presentes na biblioteca *scikit learning*.

A matriz de confusão permite a medição do modelo, relacionando os valores reais com os valores preditos [9]. Trata-se de uma métrica importante na avaliação de classificadores, por ser simples de analisar.

Sendo denotada como uma matriz quadrada, onde as diagonais principais são os números de acerto para cada classe. Os demais elementos que a compõem são os erros obtidos durante o processo de classificação, ou seja, indicam o quanto o modelo errou no momento de realizar dada classificação.

Matematicamente, define-se o seguinte:

$$\mathbf{M}(ci, cj) = \sum |f(x) = Cj| \quad (1)$$

Onde f(x) é o valor obtido pelo modelo e *cj* é o j-ésimo valor verdadeira da classe.

Pode-se então montar a seguinte matriz que relaciona todas as classes possíveis e seus possíveis resultados.

Tabela 1: Matriz de classificação

Classe	Classe 1 predita	Classe 2 predita	...	Classe N predita
Classe 1	**M** (C1, C1)	**M** (C1, C2)	...	**M** (C1, CN)
Classe 2	**M** (C2, C1)	**M** (C2, C2)	...	**M** (C2, CN)
...	...	...	...	...
Classe N	**M** (CN, C1)	**M** (CN, C1)	...	**M** (CN, CN)

Portanto, tal conceito teve maior utilização na avaliação dos modelos desenvolvidos para a resolução de problemas de classificação.

A precisão trata-se do grau de variação dos resultados de uma medição [10]. Em outras palavras, a precisão pode indicar o quanto modelo errou, baseado nos valores esperados e nos

valores obtidos. Servindo, portanto, como medição de performance de um modelo. No presente trabalho foi utilizada com essa finalidade, para tal foi usada a função “classification_report”, presente na biblioteca *scikit learning*.

4 Processo de desenvolvimento

O conjunto de ferramentas dispostos por esse conceito é vasto. Seu conhecimento é de vital importância, visto que cada um resolve um tipo de problema com maior performance. É importante conhecer bem o modelo já que o mesmo possui um papel importante na resolução do problema.

O processo de aplicação dos métodos consiste nas etapas descritas no diagrama a presentado na Figura 1:

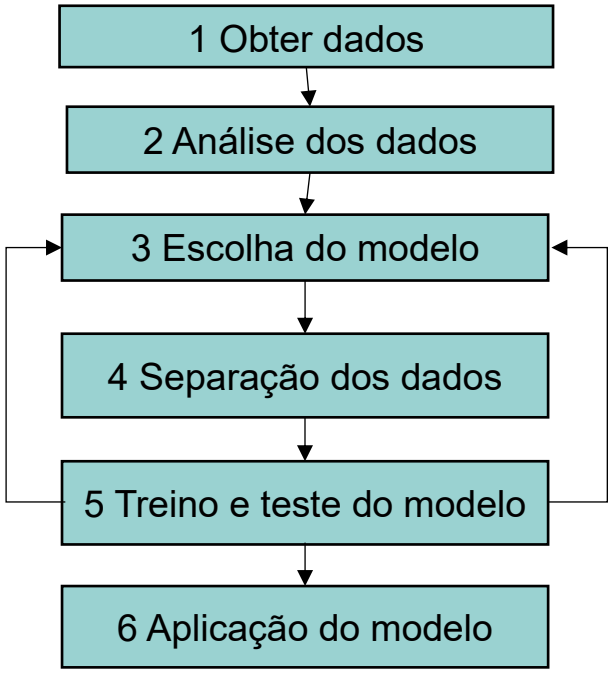

Figura 1: Diagrama do processo adotado

A partir dessa Figura, foi desenvolvido o modelo para classificação.

4.1 Obter dados

Trata-se da etapa da obtenção dos dados. Podendo ser obtidos a partir de experimentos, banco de dados públicos, observações de fenômenos naturais, entre outros. Os dados aqui utilizados estão presentes na biblioteca *Seaborn*. Podendo ser obtidos conforme a Figura 2, primeiramente a biblioteca foi importada com o comando “import seaborn as sns”, após foi

criada uma variável do tipo *DataFrame* denominada “íris”, e o *dataset* foi então carregado com a função “load_dataset(‘iris’)”.

```
# Importando o dataset
import seaborn as sns
iris = sns.load dataset('iris')
```

Figura 2: Obtenção dos dados

Um *DataFrame* é uma estrutura de dados em Python, bastante similar a uma planilha eletrônica, foi criada de forma a facilitar a análise de dados. A mesma foi utilizada para armazenamento e manipulação dos dados. Os dados são organizados em linhas e colunas. A organização dos dados utilizados se encontra na Figura 3. Onde Comprimento das sépalas é representado como *sepal_length*, Largura das sépalas é representado como *sepal_width*, Comprimento das pétalas é representado como *petal_length* e Largura das pétalas é representado como *petal_width*.

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa

Figura 3: Obtenção dos dados

43. Análise dos dados

Após os dados serem obtidos, torna-se necessário analisá-los, para buscar determinar as variáveis que melhor descrevem o problema. O uso de *Application Programming Interface* (Interface de programação de aplicativos), torna o processo mais simplificado, porém torna-se importante seu conhecimento para que o modelo desenvolvido possa obter maior desempenho. A análise consistiu na verificação de que se existiam dados faltantes e se os mesmos estavam normalizados, isto é, se os valores estavam em uma escala consistente.

Na linguagem de programação Python um dado faltante é representado pelo valor especial NaN. Esses valores podem impactar negativamente no desempenho do modelo. Para verificação da existência de algum campo sem preenchimento, primeiramente foi importada a biblioteca *pandas* que permite a manipulação de *DataFrames*. Após aplicou-se a função

“isnull(dados)” que verifica se existem valores NaN no DataFrame, retornando verdadeiro caso haja um dado faltante em um campo ou falso caso contrário. Aninhada com tal função utilizou-se a função “sum()” que soma diversos valores. O procedimento pode ser visto na Figura 4.

```
import pandas as pd
pd.isnull(iris).sum()
```

Figura 4: Obtenção dos dados

A função “sum()”, é aplicada em cada coluna do *DataFrame*. Na linguagem de programação Python, os valores verdadeiros, denotados por *True* são representados por 1 e os valores falsos, denotados por *False* são representados por 0. Portanto se existir ao menos um valor faltante o retorno será maior que 1. Se o valor for maior que 1, cabe então ao cientista analisar a melhor escolha para poder tratar aquele dado faltante. Como por exemplo: Adicionar a média dos valores anteriores. O resultado obtido se encontra na Figura 5. Como não houve nenhum resultado maior que 1, não é necessário aplicar nenhuma técnica para tratar esses dados faltantes.

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Figura 5: Resultado obtido

Outra forma empregada para analisar os dados foi traçar os gráficos de todas as variáveis que compõem o *DataFrame* duas a duas. Em outras palavras, um plano cartesiano bidimensional, é composto por dois eixos, o eixo das abscissas representado pela letra x e o eixo das ordenadas representado pela letra y, a metodologia consistiu em construir uma matriz $n \, x \, n$, onde n é a quantidade de variáveis que contém todas as possíveis combinações das variáveis duas a duas. Por exemplo, em um gráfico a variável “sepal_length” é o eixo x e a variável “species” é o eixo y, em outro ocorre o contrário isso para todas as variáveis. Vale a ressalva que também foi plotado o gráfico de uma variável com ela mesma, tais gráficos se encontram nas diagonais da matriz.

Para tanto, foi utilizada a função “pairplot(dados, separação)” da biblioteca *Seaborn* para obtenção dos gráficos, onde dados refere-se ao *dataset* de íris aqui representado pela variável “iris” e separação se refere a separação em termos de cor aplicada nos pontos que

compõem os gráficos, a separação a partir da variável “espécies”, cada cor representa uma espécie diferente.

```
# Plotando todos os gráficos possiveis com as diversas combinações das variaveis
# presentes no data set
sns.pairplot(iris, hue ='species')
plt.show()
```

Figura 6: Obtenção dos dados

A matriz de gráficos obtida, encontra-se representada na Figura 7. A partir de uma análise empírica, pode-se verificar que os dados possuem certa separação. Os pontos de coloração azul se encontram isolados enquanto os pontos de cores laranja e verde se encontram próximos, porém são de fácil separação.

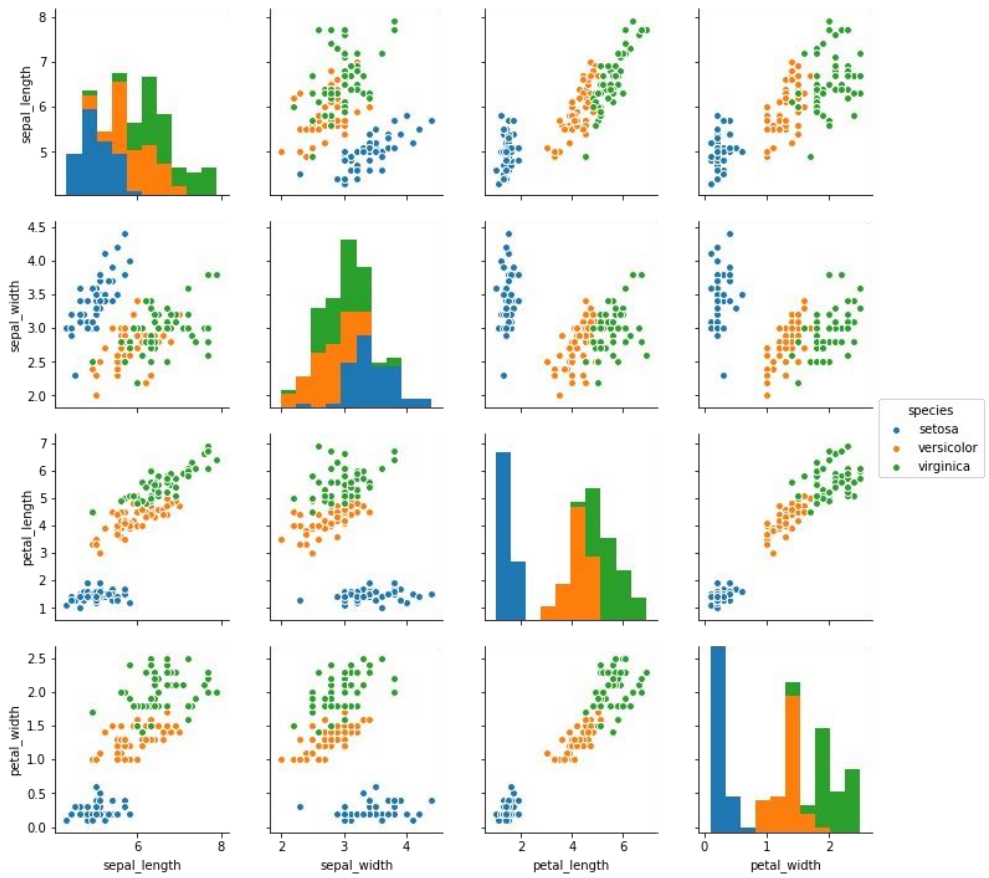

Figura 7: Matriz de gráficos

Portanto, torna-se evidente que o problema de classificar uma flor de íris pode ser resolvido como um problema de classificação, utilizando técnicas de *machine learning*. Buscando assim, tornar tal, mais rápida, precisão e realizada de forma automatizada. Cabendo ao ser humano apenas a tarefa de avaliar a resposta.

4.2 Escolha do modelo

Após escolhido os dados a serem utilizados, torna-se necessário escolher o modelo que melhor possa se ajustar aos dados, bem como seja capaz de executar seu proposito perfeitamente e possuir uma boa performance.

O modelo escolhido foi o *k Nearest Neighbors* já que o mesmo é aplicado em problemas de classificação, podendo também, ser aplicado em problemas de regressão [11], porém com menor recorrência. O modelo baseia-se na proximidade que os dados possuem uns dos outros. Sendo, simples e possuindo poucos parâmetros, porém apresenta alto custo computacional [1].

O modelo utilizado se encontra na biblioteca *scikit learning* representada por *sklearn*. O seu uso, permite maior eficiência na procura por soluções, já que não é necessário a implementação do método em si, apenas seu uso.

Na Figura 8 é possível verificar a importação bem como a instanciação do modelo que será utilizado para a classificação. A importação é representada pela segunda linha “from sklearn.neighbors import KneighborsClassifier” e a instanciação representada pela terceira linha “knn = KNeighborsClassifier(n_neighbors=1)”. Onde n_neighbors=1, representada a quantidade de vizinhos de um ponto, ou seja, apenas um ponto foi escolhido para classificar a qual classe outro ponto pertence. O valor escolhido foi 1, na seção “Escolha do valor de K” será explicado o porquê.

```
# Importando o classificador que reside na biblioteca Scikit Learning
from sklearn.neighbors import KneighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

Figura 8: Importação e instanciação do classificador

4.3 Separação dos dados

Etapa essa aplicada na aprendizagem supervisionada, os dados são separados em dois conjuntos, preferencialmente de tamanhos distintos. O conjunto de dados maior é utilizado para treinamento, o conjunto menor é utilizado para testar o modelo. Houve a separação dos dados buscando evitar o *overfitting*.

A separação consistiu em usar 30 % dos dados para testes e 70 % para treinar o modelo. Sendo utilizado, o objetivo é evitar o *overfitting*, que faz com que o modelo se torne muito

adequado apenas para os dados de treino, já que o modelo não é testado. Para a separação foi utilizada a função “train_test_split”, que separa os dados, baseados em uma distribuição normal.

As variáveis utilizadas no treino são duas, as Preditoras, que são as variáveis utilizadas para classificar a espécie de íris e as variáveis que representam os rótulos, ou seja, as respostas. No presente trabalho as características: comprimento e largura das sépalas e comprimento e largura das pétalas serão as variáveis preditores, representada pela letra X e a espécie será o rotulo representada pela letra y.

Sendo assim, para a separação dos dados foi utilizada o trecho de código presente na Figura 9. Inicialmente foi importada a função, presente na primeira linha, após as variáveis foram separadas, utilizando manipulações simples de *DataFrames*. A variável X recebeu todas as partes do *DataFrame* exceto a coluna *species* que pertence a variável y. Na última linha foi aplicada a função em si, X se refere a variável X e y se refere a variável y, “random_state” é um parâmetro opcional utilizado apenas para que outras execuções gerem resultados semelhantes. O último parâmetro “test_size” serve para indicar a porcentagem dos dados que ficará para o treino, no caso, 30 % ficaram para o treino.

As variáveis “X_train”, “X_test”, “y_train”, “y_test”, são as variáveis já separadas para treino e teste.

```
from sklearn.model_selection import train_test_split
X = iris.drop('species', axis = 1)
y = iris['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 101, test_size = 0.3)
```

Figura 9: Trecho de separação dos dados

4.4 Treino e teste do modelo

O treino serve para determinar um modelo que melhor se adeque ao problema. O treino foi realizado com apenas uma linha, mostrando, portanto, a importância da linguagem de programação Python no desenvolvimento de métodos de *machine learning*. O procedimento de treino pode ser visto na Figura 10. A função que permite o treino é a função “fit(variável preditora, rotulo)” onde, variável preditora se refere a variável X e o rotulo se refere a variável y.

```
knn.fit(X_train, y_train)
```

Figura 10: Treino do modelo

A próxima etapa é realizar o teste do modelo para determinar sua acurácia, ou seja, verificar o quão boas estão suas respostas baseado nas entradas. Para realizar o teste, foi feita a predição dos valores da variável “X_test”, com a função “predict(X_test)”, conforme a Figura 11. Os valores obtidos foram armazenados em **predito**.

```
# Predizendo os valores do subconjunto de teste
predito = knn.predict(X_test)
```

Figura 11: Predição dos valores do subconjunto de teste

Ao fim, obteve-se as espécies a qual aqueles dados pertencem, baseado nas características dos mesmos. O resultado pode ser visualizado na Figura 12, sendo a saída um vetor com 30 % dos dados e o nome de cada espécie.

```
array(['setosa', 'setosa', 'setosa', 'virginica', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'virginica', 'setosa',
       'virginica', 'setosa', 'setosa', 'virginica', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'setosa',
       'virginica', 'versicolor', 'virginica', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'versicolor',
       'versicolor'], dtype=object)
```

Figura 12: Vetor de saída

A saída foi então utilizada para medir o quão bem o modelo conseguiu predizer a qual espécie de íris pertencia, bem como avaliar a performance do modelo. Para tanto, foi utilizada as funções “classification_report” (reporte de classificação, no presente trabalho a ênfase será na coluna precisão) e a “confusion_matrix” (matriz de confusão) para realizar tal medição. O resultado obtido se encontra na Figura 13, a partir de sua análise, percebe-se que o modelo obteve bons resultados, já que, não errou nenhuma vez em suas predições. Visto que, a precisão obtida foi de 1, ou seja, 100% e na matriz de confusão apenas os elementos pertencentes a diagonal principal contém valores diferentes de zero, pode se dizer que os resultados foram satisfatórios.

```
from sklearn.metrics import classification_report, confusion_matrix

print (confusion_matrix(y_test, predito))
print (classification_report(y_test,predito))
```

```
[[13  0  0]
 [ 0 20  0]
 [ 0  0 12]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      1.00      1.00        20
   virginica       1.00      1.00      1.00        12

   micro avg       1.00      1.00      1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Figura 13: Avaliação do modelo

4.5 Aplicação do modelo

Por fim a etapa de aplicação do modelo consiste em utilizar o mesmo para resolver o problema ao qual foi construído. A aplicação do modelo desenvolvido foi na classificação das flores de íris conforme sua espécie.

O modelo desenvolvido obteve uma boa performance, conseguindo predizer corretamente a espécie que a flor de íris pertence com 100% de certeza.

5 Escolha do valor de *k*

O valor de *k* escolhido foi 1. A metodologia utilizada para tal escolha, foi encontrar um valor de *k* que minimize o erro médio entre o valor predito e o valor real pelo modelo. Para tal, foram executadas as etapas de treino e predição no intervalo entre 1 e 9, foi então calculado o erro médio entre o valor predito e o real. Ao final obteve-se a Tabela 2 que contém os valores de k e os valores dos erros.

Tabela 2: Relação entre os valores de k e o erro médio

k	*erro médio*
1	0
2	0. 044444444444444446
3	0
4	0.044444444444444446
5	0
6	0
7	0.022222222222222223
8	0.044444444444444446
9	0

Como pode-se observar o valor de 1, assim como outros, resultou em um erro médio de 0, ou seja, não existiu erro para aquela predição. Sendo assim, o valor de $k = 1$ foi escolhido, devido, apresentar taxa de erro igual a 0. Somado a este fato, com tal valor, necessita-se apenas de um ponto para realizar a classificação impactando na performance do modelo. Para o cálculo dos modelos, foi utilizado o seguinte trecho de código, presente na Figura 14.

```
# Econtrando o melhor k
taxa_erro = []

for k in range(1, 10):
    knn =KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    predito = knn.predict(X_test)
    taxa_erro.append(np.mean(predito !=y_test))
```

Figura 14: Trecho de código para encontrar o melhor valor de k

6 Conclusão

Ao final da execução dos métodos, bem como de todos os passos descritos, obteve-se um ótimo modelo. Com isso, os resultados esperados foram alcançados com sucesso. Tal fato se deu devido ao modelo apresentar 100% de precisão, sendo, portanto, um bom modelo para classificação do *dataset* de íris. A biblioteca *scikit learning* mostrou-se uma ferramenta valiosa para estudo e aplicação de técnicas de *machine learning*. Visto que, a mesma facilita a implementação dos métodos, cabe ao cientista apenas a tarefa de análise e ajuste dos métodos.

7 Referências Bibliográficas

[1] JAMES, Gareth et al. **An Introduction to Statistical Learning with Applications in R**. New York: Springer, 2013.

[2] FARINACCIO, Rafael. **Inteligência artificial é capaz de reconhecer até rostos disfarçados**. Disponível em: <https://www.tecmundo.com.br/ciencia/121836-inteligencia-artificial-capaz-reconhecer-rostos-disfarcados.htm>. Acesso em: 20 nov. 2018.

[3] GHOSH, Pallab. **Inteligência artificial pode levar ao diagnóstico precoce de doenças cardíacas e câncer de pulmão**. Disponível em: <https://www.bbc.com/portuguese/geral-42537252>. Acesso em: 3 jan. 2019.

[4] HONDA, Hugo; FACURE, Matheus; YAOHAO, Peng. **Os Três Tipos de Aprendizado de Máquina**. Disponível em: <https://lamfo-unb.github.io/2017/07/27/tres-tipos-am/>. Acesso em: 11 nov. 2018.

[5] MENEZES, Paulo. **Aprendizagem Supervisionada e Aprendizagem não Supervisionada**. Disponível em: <https://home.isr.uc.pt/~paulo/PROJ/NN95/node31.html>. Acesso em: 25 out. 2018.

[6] LIMA, Edirlei Soares de. **Aprendizado Não-Supervisionado**. Rio de Janeiro: Visual, 2011. Color. Disponível em: <http://edirlei.3dgb.com.br/aulas/ia_2011_2/IA_Aula_18_Aprendizado_Nao_Supervisionado.pdf>. Acesso em: 09 dez. 2018.

[7] PINHEIRO, Nilcéia Aparecida Maciel. **Uma reflexão sobre a importância do conhecimento matemático para a ciência, para a tecnologia e para a sociedade**. Letras e Arte, Ponta Grossa, v. 11, n. 22, p.22-31, abr. 2003.

[8] SHIMAKURA, Silvia Emiko. **Tipos de variáveis**. Disponível em: <http://leg.ufpr.br/~silvia/CE055/node8.html>. Acesso em: 20 set. 2012.

[9] MICROSOFT. **Matriz de classificação (Analysis Services - Mineração de dados)**. Disponível em: <https://docs.microsoft.com/pt-br/sql/analysis-services/data-mining/classification-matrix-analysis-services-data-mining?view=sql-server-2017>. Acesso em: 30 abr. 2018.

[10] BALBINOT, Alexandre; BRUSAMARELLO, Valner João. **Instrumentação e Fundamentos de Medida**. 2. ed. Rio de Janeiro: Ltc, 2010.

[11] GANDHI, Rohith. **K Nearest Neighbours—Introduction to Machine Learning Algorithms**. Disponível em: <https://towardsdatascience.com/k-nearest-neighbours-introduction-to-machine-learning-algorithms-18e7ce3d802a>. Acesso em: 15 nov. 2018.

[12] SCIKIT-LEARN. **Scikit-learn Machine Learning in Python.** Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 20 out. 2018.