

ISABEL SALES DE CASTRO ALMEIDA

DRE:118016977

## Support Vector Machine (SVM)

**Resumo:** Este documento corresponde a parte 2 do Trabalho Prático III do curso de Inteligência Computacional II (CPS849) do Programa de Engenharia de Sistemas e Computação da Coppe - UFRJ. O mesmo visa implementar um experimento computacional com uso do algoritmo Support Vector Machine (SVM) em uma conhecida base de testes disponibilizada pela Knowledge Extraction Evolutionary Learning (KEEL)..

**Palavras-chave:** Aprendizado de Máquina, Classificação, Máquinas de Vetores de Suporte (Support Vector Machines)

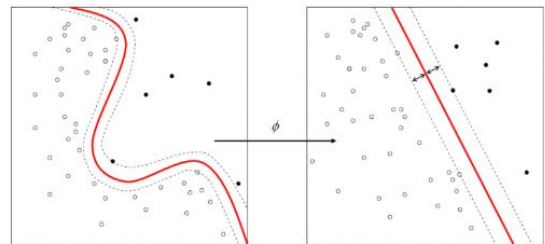
### I. INTRODUÇÃO

Support Vector Machines (SVMs) é um método popular de aprendizado de máquina para classificação, regressão e outras tarefas de aprendizagem. As SVMs foram introduzidas inicialmente na década de 1960 e foram posteriormente refinadas na década de 1990. Contudo, agora elas têm se tornando extremamente populares, devido à sua capacidade de alcançar ótimos resultados.

As SVMs são embasadas pela teoria de aprendizado estatístico, desenvolvida por Vapnik [2] a partir de estudos iniciados em [3]. Essa teoria estabelece uma série de princípios que devem ser seguidos na obtenção de classificadores com boa generalização, definida como a capacidade de prever corretamente a classe de novos dados do mesmo domínio em que o aprendizado ocorreu.

Dado um conjunto de treinamento, cada um pertencente a uma categoria específica, um algoritmo de treinamento de **SVM** cria um

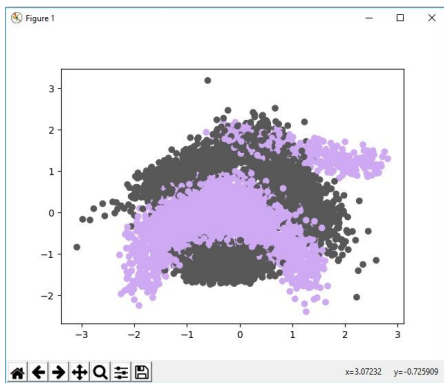
modelo que separa as categorias e que posteriormente pode ser usado para decidir a categoria do novo conjunto de dados.



O método consiste em encontrar o melhor plano de separação (com margem máxima) baseado em um vetor específico chamado **vetor de suporte**. Caso a decisão não for viável no espaço de descrição inicial, é possível aumentar a dimensão do espaço graças às funções do **kernel** e também é possível encontrar um hiperplano que será o separador de decisão.

### II. EXPERIMENTO COMPUTACIONAL

O experimento objetiva averiguar a capacidade do SVM na classificação de uma base de dados sintética da KEEL. Conhecida como Base Banana, em que as instâncias pertencem a vários clusters que se apresentam com formato de banana, tal base apresenta duas classes (1) e (-1) para um conjunto de 5300 observações contendo 2 atributos, At1 e At2 correspondentes aos eixos x e y, respectivamente.



Na figura acima, temos um gráfico de dispersão, os pontos roxos estão na classe 1 e os pontos cinzas estão na classe -1. O desafio é encontrar um separador que possa indicar se um novo dado está na banana ou não.

### III . OTIMIZAÇÃO DO MODELO

A otimização de SVM é um processo iterativo que visa maximizar a margem, dependendo dos vetores de suporte escolhidos.

$$\text{Maximiser } \tilde{L}(\alpha) = \sum_{k=1}^p \alpha_k - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j l_i l_j x_i^T x_j$$

Com o multiplicador de Lagrange ideal  $\alpha$ , a função de decisão está completa. A equação do hiperplano pode ser traduzida no espaço de descrição inicial, graças à função do kernel k, da seguinte forma:

$$f(x) = \sum_{i=1}^p \alpha_i l_i k(x, x_i) + b$$

No exemplo, temos:

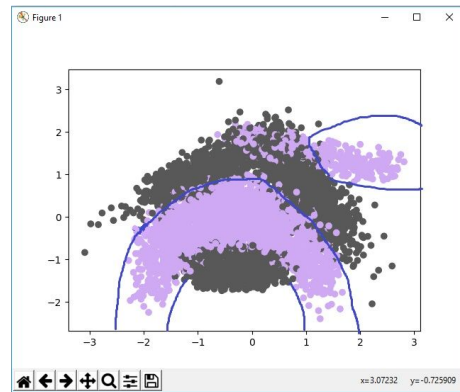
$$f(x) = \sum_{i=1}^{nSV} \alpha_i \cdot l_i \cdot \exp(-\gamma \|x - x_i\|^2) - \rho$$

com:

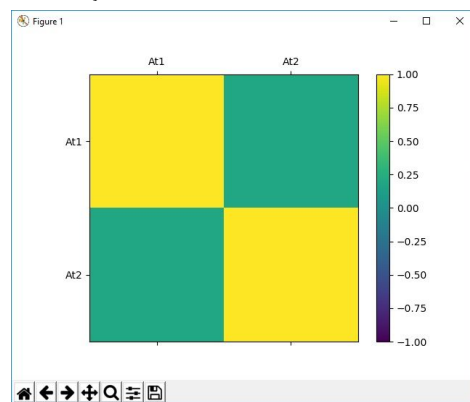
- nSV: número de vetores de suporte
- $\alpha_i \cdot l_i = \text{sv\_coef}$
- $x_i$ : vetores de suporte SVs
- $\rho = -b$

Quando a função de decisão para um determinado dado é positiva, então esses dados

pertencem à classe da banana (roxo). Caso contrário, pertence à classe cinza.

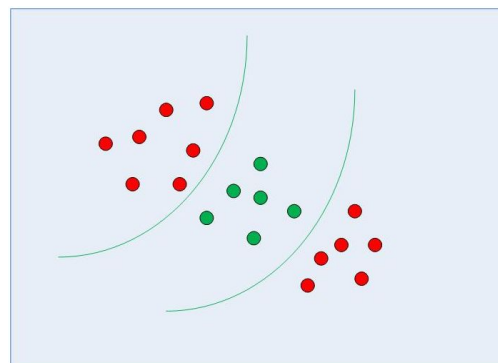


Abaixo temos At1 e At2 na matriz de correlação:



### IV . KERNEL SVM

Para o caso de dados não linearmente separáveis, como na figura abaixo, uma linha reta não pode ser usada como um limite de decisão.



No caso de dados não linearmente separáveis, o algoritmo SVM simples não pode ser usado. Em vez disso, uma versão modificada do SVM, chamada Kernel SVM, é usada. Algoritmos SVM usam um conjunto de

funções matemáticas que são definidas como o kernel. A função do kernel é pegar dados como entrada e transformá-los no formulário requerido. Diferentes algoritmos de SVM usam diferentes tipos de funções do kernel. Essas funções podem ser de tipos diferentes, como por exemplo, função de base linear, não linear, polinomial, radial (RBF) e sigmóide.

É possível introduzir funções do Kernel para dados de sequência, gráficos, texto, imagens e vetores. O tipo mais usado de função do kernel é o RBF. Porque tem resposta localizada e finita ao longo de todo o eixo x.

As funções do kernel retornam o produto interno entre dois pontos em um espaço de recurso adequado. Assim, definindo uma noção de similaridade, com pouco custo computacional, mesmo em espaços de dimensões muito altas.

## V. EXEMPLOS DE KERNEL

A função kernel, como dito anteriormente, é usada para aumentar a dimensão espacial. Aqui estão alguns tipos mais comuns de kernel e suas respectivas funções:

- **Linear:**

$$K(u, v) = u' \cdot v$$

- **Polinomial:**

$$K(u, v) = (\gamma u' \cdot v + coef_0)^{degree}$$

- **Função de Base Radial (RBF):**

$$K(u, v) = \exp(-\gamma \cdot |u - v|^2)$$

- **Sigmóide:**

$$K(u, v) = \tanh(\gamma u' \cdot v + coef_0)$$

## VI. COMPARAÇÃO DO DESEMPENHO COM K-FOLD CROSS VALIDATIONS

Foi utilizado a biblioteca Scikit-Learn, onde o *train\_test\_test* e o *cross\_val\_score* da validação cruzada foram importados. Em seguida, foi dividido o conjunto de dados

original em conjunto de dados de treinamento e teste para  $k = 2, 5$  e  $10$ .

**Para  $k = 2$ :**

	precision	Eout
<b>Kernel Gaussiano (RBF)</b>	0.9037735849056603	0.09622641509433971
<b>kernel polinomial</b>	0.629245283018868	0.37075471698113205
<b>kernel sigmoid (gamma=1)</b>	0.2818867924528302	0.7181132075471698
<b>kernel sigmoid (gamma=0.5)</b>	0.2922641509433962	0.7077358490566038
<b>kernel sigmoid (gamma=0.01)</b>	0.5516981132075471	0.44830188679245286
<b>kernel linear</b>	0.5516981132075471	0.44830188679245286

**Para  $k = 5$ :**

	precision	Eout
<b>Kernel Gaussiano (RBF)</b>	0.9024563562569584	0.09754364374304159
<b>kernel polinomial</b>	0.6267900063453444	0.37320999365465557
<b>kernel sigmoid (gamma=1)</b>	0.28188422136837343	0.7181157786316266
<b>kernel sigmoid (gamma=0.5)</b>	0.29263876186137516	0.7073612381386248
<b>kernel sigmoid (gamma=0.01)</b>	0.5516981315112639	0.4483018684887361
<b>kernel linear</b>	0.5516981315112639	0.4483018684887361

**Para  $k = 10$ :**

	precision	Eout
<b>Kernel Gaussiano (RBF)</b>	0.902077686285818	0.09792231371418203
<b>kernel polinomial</b>	0.625084730313662	0.37491526968633804
<b>kernel sigmoid (gamma=1)</b>	0.28112540081048404	0.718874599189516
<b>kernel sigmoid</b>	0.292066333921	0.707933666078

(gamma=0.5)	0018	9982
kernel sigmoid (gamma=0.01)	0.551698260981067	0.448301739018933
kernel linear	0.551698260981067	0.448301739018933

É possível notar que apenas o Kernel Sigmóide com  $\gamma = 0.01$  apresenta melhor desempenho para  $k = 10$  e menor para  $k = 2$ . Para os demais Kernels, o melhor desempenho está em  $k = 2$  e o pior para  $k = 10$ . Mas a diferença de desempenho é mínima, sendo apenas possível a verificação devido ao grande número de casas decimais utilizadas no experimento.

## VII. COMPARAÇÃO DO DESEMPENHO KERNEL

Para treinar o kernel SVM, foi usada a classe SVC da biblioteca do Scikit-Learn SVM. A diferença está no valor do parâmetro do kernel da classe SVC. No caso do SVM simples, usamos "linear" como o valor para o parâmetro do kernel. No entanto, para o SVM do kernel, pode-se usar o kernel RBF, polinomial, sigmoid ou computable. Foram implementados núcleos polinomiais, gaussianos (RBF) e sigmóides para ver qual deles apresenta melhor desempenho.

### Kernel Linear

	precision	recall	f1-score	support
-1	0.54	1.0	0.70	571
1	0.00	0.00	0.00	489
avg/total	0.29	0.54	0.38	1060

### Kernel Polinomial

	precision	recall	f1-score	support
-1	0.60	0.99	0.75	599
1	0.92	0.16	0.27	461
avg/total	0.74	0.63	0.54	1060

### Kernel Sigmoid (gamma=1)

	precision	recall	f1-score	support
-1	0.32	0.33	0.32	567
1	0.19	0.18	0.18	493
avg/total	0.26	0.26	0.26	1060

### Kernel Sigmoid (gamma=0.5)

	precision	recall	f1-score	support
-1	0.35	0.34	0.35	588
1	0.21	0.22	0.22	472
avg/total	0.29	0.29	0.29	1060

### Kernel Sigmoid (gamma=0.01)

	precision	recall	f1-score	support
-1	0.56	1.00	0.72	595
1	0.00	0.00	0.00	465
avg/total	0.32	0.56	0.40	1060

### Kernel Gaussiano (RBF)

	precision	recall	f1-score	support
-1	0.89	0.92	0.90	587
1	0.90	0.85	0.87	473
avg/total	0.89	0.89	0.89	1060

Comparando o desempenho dos diferentes tipos de kernels, podemos ver claramente que o kernel sigmóide é o que apresenta o pior

desempenho para diferentes valores de  $\gamma$  (1, 0.5 e 0.01), embora quanto menor seja o valor de  $\gamma$ , melhor será o seu desempenho. Isso se deve ao fato da função sigmóide retornar dois valores, 0 e 1, portanto, é mais adequado para problemas de classificação binária.

O kernel gaussiano (RBF) e o núcleo polinomial apresentaram os melhores desempenhos. No entanto, não existe uma regra rígida quanto ao melhor desempenho do

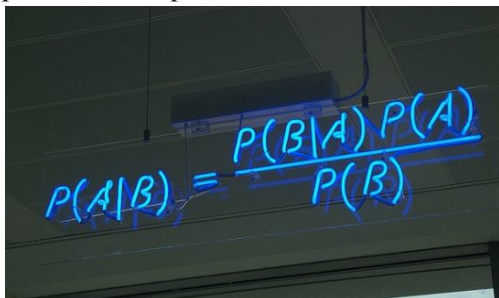
kernel em todos os cenários. É necessário testar todos os kernels e selecionar aquele com os melhores resultados para o conjunto de dados de teste. Neste conjunto de dados de teste, o melhor desempenho é o do Kernel Gaussiano (RBF).

## VIII. MODELOS DE CLASSIFICAÇÃO

Foram escolhidos cinco modelos de classificação para que fosse possível comparar os seus desempenhos com o Kernel Gaussiano (RBF) que apresentou melhor resultado comparado aos demais Kernels.

### Naïve Bayes - Gaussian:

Os classificadores Naïve Bayes são uma família de classificadores probabilísticos simples com base na aplicação Bayes 'teorema com forte independência entre as características. A imagem é a equação – em que  $P(A|B)$  é a probabilidade posterior,  $P(B|A)$  é a probabilidade,  $P(A)$  é a probabilidade prévia e  $P(B)$  é preditor de probabilidade prévia.



$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

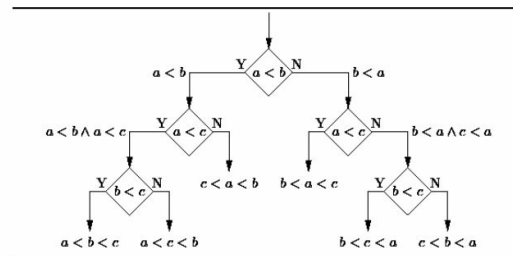
A escolha desse método se deve ao fato deste poder ser utilizados em diversos casos, como por exemplo:

- Para marcar um e-mail como spam ou não spam
- Classificar um artigo de notícias sobre tecnologia, política ou esportes
- Verificar um pedaço de texto expressando emoções positivas ou negativas
- Usado para software de reconhecimento facial

### Decision Tree:

Uma árvore de decisão é uma ferramenta de apoio que utiliza um gráfico ou modelo de decisões e suas possíveis consequências,

incluindo resultados de eventos fortuitos, custos de recursos e utilidade.



Do ponto de vista da decisão de negócios, uma árvore de decisão é o número mínimo de perguntas que devem ser respondidas para avaliar a probabilidade de tomar uma decisão correta, na maioria das vezes. Esse método foi escolhido, por permitir abordar o problema de uma forma estruturada e sistemática para chegar a uma conclusão lógica.

### K-nearest neighbors:

$K$ -NN é um tipo de aprendizagem baseada em instância, ou aprendizagem preguiçoso, onde a função só é aproximada localmente e toda computação é adiada até a classificação. O algoritmo  $k$ -NN está entre os mais simples de todos os algoritmos de aprendizado de máquina, por isto a sua escolha para este experimento.

A entrada consiste nos  $k$  exemplos de treinamento mais próximos no espaço de recursos. A saída depende se  $k$ -NN é usada para a classificação ou regressão:

- Na *classificação k-NN*, a saída é uma associação de classe. Um objeto é classificado pelo voto da maioria de seus vizinhos, com o objeto que está sendo atribuído à classe mais comum entre seus  $k$  vizinhos mais próximos ( $k$  é um positivo inteiro, tipicamente pequenos). Se  $k = 1$ , então o objeto é simplesmente atribuído à classe daquele único vizinho mais próximo.
- Na *regressão k-NN*, a saída é o valor da propriedade para o objeto. Este valor é a média dos valores de seus  $k$  vizinhos mais próximos.

**Bagging:**

É uma técnica que combina as previsões de vários algoritmos de aprendizado de máquina para fazer previsões mais precisas do que qualquer modelo individual. É um procedimento geral que pode ser usado para reduzir a variação para aqueles algoritmos com alta variância. Assim como as próprias árvores de decisão, o Bagging pode ser usado para problemas de classificação e regressão, por isto, sua inclusão no experimento.

**Random Forest:**

É um método de aprendizado conjunto para classificação, regressão e outras tarefas que operam construindo uma multiplicidade de árvores de decisão no momento do treinamento. Random Forest corrige o hábito de overfitting de árvores de decisão para seu conjunto de treinamento, de modo que as previsões resultantes de todas as subárvores tenham menos correlação, sendo assim, um dos métodos escolhidos para comparação.

**IX. COMPARAÇÃO DO DESEMPENHO**

Abaixo segue um comparativo de desempenho entre os diferentes métodos apresentados:

	<b>Precision</b>
<b>Kernel Gaussiano (RBF)</b>	0.9031446540880503
<b>Naive Bayes - Gaussian</b>	0.60062893081761
<b>Decision Tree</b>	0.869811320754717
<b>K-nearest neighbors</b>	0.8981132075471698
<b>Bagging</b>	0.8974842767295598
<b>Random Forest</b>	0.9018867924528302

Deste modo, é possível concluir que o SVM com o Kernel Gaussiano (RBF), supera os outros algoritmos, além disso é possível aumentar a precisão ajustando o hiperparâmetro dos algoritmos usados.

**X. CONCLUSÃO**

Neste projeto foi apresentado o SVMs simples e o de kernel. Como o algoritmo SVM funciona e os resultados obtidos por meio da sua implementação com a biblioteca Scikit-Learn do Python. Também foi apresentado os diferentes tipos de kernels que podem ser usados para implementar o SVM Kernel e um comparativo de desempenho entre eles, por meio da utilização da base de dados sintética da KEEL, conhecida como Base Banana. A implementação deste trabalho está disponível no anexo e em:

<https://github.com/isabelsales/Trabalho-pratico-III/blob/master/svm.py>

**XI. REFERÊNCIAS**

- [1] Implementing SVM and SVM Kernel with Python's Scikit-Learn, Disponível em: <<https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/>>. Acessado em 29 de agosto de 2018.
- [2] V. N. Vapnik. The nature of Statistical learning theory. Springer-Verlag, New York, 1995.
- [3] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications, 16(2):283–305, 1971.
- [4] Lecture 14 - Support Vector Machine, Disponível em: <<https://www.youtube.com/watch?v=MEG35RDD7RA>>. Acessado em 29 de agosto de 2018.
- [5] Machine learning - Classification with SVM, Disponível em: <<https://scilab.io/machine-learning-classification-with-svm/>>. Acessado em 29 de agosto de 2018.
- [6] KEEL-dataset, Disponível em: <<http://sci2s.ugr.es/keel/dataset.php?cod=182>>. Acessado em 29 de agosto de 2018.
- [7] SUPPORT VECTOR MACHINE (SVM CLASSIFIER) IMPLEMENTATION IN PYTHON WITH SCIKIT-LEARN, Disponível em: <<http://dataaspirant.com/2017/01/25/svm-classifier-implementation-python-scikit-learn/>>. Acessado em 29 de agosto de 2018.
- [8] Kernel Functions, Disponível em: <<https://data-flair.training/blogs/svm-kernel-functions/>>. Acessado em 29 de agosto de 2018.

[9] Bagging and random forest ensemble algorithms for machine learning, Disponível em:

<<https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/>>. Acessado em 29 de agosto de 2018.

[10] K-nearest neighbors algorithm, Available in:

<[https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)>. Acessado em 29 de agosto de 2018.

[11] Random forest, Disponível em:

<[https://en.wikipedia.org/wiki/Random\\_forest](https://en.wikipedia.org/wiki/Random_forest)>.

Acessado em 29 de agosto de 2018.

[12] 10 algoritmos de machine learning, Disponível em:

<<http://www.semantix.com.br/blog/10-algoritmos-d>

e-machine-learning/>. Acessado em 29 de agosto de 2018.

[13] `learn.kernel_ridge.KernelRidge`, Disponível em:

<[http://scikit-learn.org/stable/modules/generated/sklearn.kernel\\_ridge.KernelRidge.html](http://scikit-learn.org/stable/modules/generated/sklearn.kernel_ridge.KernelRidge.html)>. Acessado em 29 de agosto de 2018.

[14] `Cross_validation`, Disponível em:

<[http://scikit-learn.org/stable/modules/cross\\_validation.html](http://scikit-learn.org/stable/modules/cross_validation.html)>. Acessado em 29 de agosto de 2018.

## XII. ANEXO

Abaixo segue o código do experimento devidamente comentado:

```
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
import matplotlib.mlab as mlab

# Quaisquer resultados gravados no diretório atual são salvos como saída.
open_file = pd.read_csv("input/banana3.csv", sep=",")
#print(open_file.head())
#print(open_file.shape)

#print(open_file.isnull().values.any())

#print(open_file.describe())

#Agora, usando a biblioteca Matplotlib, plotando os dois recursos no gráfico de dispersão

import matplotlib.pyplot as plt

x = open_file[["At1"]]
y = open_file[["At2"]]
c = open_file[["Class"]]

for i in c:
    col = np.where(c[i]==-1, '#585858', '#D0A9F5')

plt.scatter(x, y, c=col)
plt.show()

#Agora, plotar os dois recursos na matriz de correlação
```



```

file = open_file[['At1','At2']]
#print(file.head())

correlation = file.corr()
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlation, vmin=-1, vmax=1)
fig.colorbar(cax)
names=['At1','At2']
ticks = np.arange(0,2,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()

#Agora, usando a biblioteca sklearn, importamos o train test test da validação
# cruzada e dividimos o conjunto de dados original em conjunto de dados de treinamento e teste

from sklearn.cross_validation import train_test_split
train_test = train_test_split(open_file,test_size=0.3)
features_train = train[['At1','At2']]
features_test = test[['At1','At2']]
labels_train = train.Class
labels_test = test.Class
print(labels_test.head())
print(train.shape)
print(test.shape)

```

```

#Agora, podemos usar nossos algoritmos de aprendizado de máquina para brincar com treinamento e conjunto de dados de teste.
# Começamos com o classificador "Naive Bayes - Gaussian"

from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
training = clf.fit(features_train,labels_train)
predictions = clf.predict(features_test)
print("Previsão: ",predictions)
print("Precisão do classificador Naive Bayes - Gaussian:",clf.score(features_test,labels_test))

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

#"Support Vector Machine" com kernel Polinomial
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
clf = SVC(kernel='poly', degree=8)
training = clf.fit(features_train,labels_train)
predictions = clf.predict(features_test)
print("Previsão: ",predictions)
print("Precisão com kernel polinomial:",clf.score(features_test,labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.0.20)
y_pred = clf.predict(X_test)

scores = cross_val_score(clf, X, y, cv=2)
total = 0

```



```

for i in scores:
    total_ += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total_ += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))
scores = cross_val_score(clf, X, y, cv=10)
total = 0
for i in scores:
    total_ += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

```

```

# "Support Vector Machine" com kernel sigmoid (gamma=1)
from sklearn.svm import SVC
clf = SVC(kernel='sigmoid', gamma=1)
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com kernel sigmoid (gamma=1): ", clf.score(features_test, labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
y_pred = clf.predict(X_test)

scores = cross_val_score(clf, X, y, cv=2)
total = 0
for i in scores:
    total_ += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total_ += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))

```

```

scores = cross_val_score(clf, X, y, cv=10)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

# "Support Vector Machine" com kernel sigmoid (gamma=0.5)
from sklearn.svm import SVC
clf = SVC(kernel='sigmoid', gamma=0.5)
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com kernel sigmoid (gamma=0.5):", clf.score(features_test, labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
y_pred = clf.predict(X_test)

scores = cross_val_score(clf, X, y, cv=2)

```

```

total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))
scores = cross_val_score(clf, X, y, cv=10)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

```

```

# "Support Vector Machine" com kernel sigmoid (gamma=0.01)
from sklearn.svm import SVC
clf = SVC(kernel='sigmoid', gamma=0.01)
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com kernel sigmoid (gamma=0.01): ", clf.score(features_test, labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
y_pred = clf.predict(X_test)

scores = cross_val_score(clf, X, y, cv=2)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))
scores = cross_val_score(clf, X, y, cv=10)

```

```

total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

# Agora, usando nosso segundo classificador como "Support Vector Machine" com kernel = "rbf"

from sklearn import svm
clf = svm.SVC(kernel='rbf')
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com kernel rbf: ", clf.score(features_test, labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
y_pred = clf.predict(X_test)

```

```

scores = cross_val_score(clf, X, y, cv=2)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))
scores = cross_val_score(clf, X, y, cv=10)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

# "Support Vector Machine" com kernel="linear".

```

```

from sklearn import svm
clf = svm.SVC(kernel='linear', C=1)
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com kernel linear:", clf.score(features_test, labels_test))

X = open_file.drop('Class', axis=1)
y = open_file['Class']

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import recall_score
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
y_pred = clf.predict(X_test)
scores = cross_val_score(clf, X, y, cv=2)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 2: ", total/2)
print("Eout: ", 1 - (total/2))
scores = cross_val_score(clf, X, y, cv=5)
total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 5: ", total/5)
print("Eout: ", 1 - (total/5))
scores = cross_val_score(clf, X, y, cv=10)

```



```

total = 0
for i in scores:
    total += i
print("Scores:", scores)
print("Score K-fold cross-validation = 10: ", total/10)
print("Eout: ", 1 - (total/10))

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

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

#Agora, usando nosso terceiro classificador como "DecisionTreeClassifier"

from sklearn import tree
clf = tree.DecisionTreeClassifier()
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão como DecisionTreeClassifier: ", clf.score(features_test, labels_test))

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

#Agora, usando nosso quarto classificador como "KNeighborsClassifier"

```

```

from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão como KNeighborsClassifier: ", clf.score(features_test, labels_test))

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

#Podemos tentar usar outro classificador - "BaggingClassifier" Um classificador do Bagging é um
# meta-estimador conjunto que se encaixa classificadores base cada um em subconjuntos aleatórios do original
# conjunto de dados e, em seguida, agregar suas previsões individuais (por votação ou por média) para formar uma previsão final

from sklearn.ensemble import BaggingClassifier
clf = BaggingClassifier(n_estimators=100, random_state=7)
boosted = clf.fit(features_train, labels_train)
prediction = clf.score(features_test, labels_test)
print("Precisão com BaggingClassifier: ", prediction)

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

#Agora, podemos tentar prever a precisão usando o RandomForestClassifier

from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=7)
boosted = clf.fit(features_train, labels_train)
prediction = clf.score(features_test, labels_test)
print("Precisão usando o RandomForestClassifier: ", prediction)

```

```

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

#Então, com isso, chegamos a saber que o SVM com o kernel 'rbf' supera outros algoritmos,
# além disso podemos aumentar a precisão ajustando o hiperparâmetro dos algoritmos usados

from sklearn import svm
clf = svm.SVC(kernel='rbf', C=10, gamma='auto')
training = clf.fit(features_train, labels_train)
predictions = clf.predict(features_test)
print("Previsão: ", predictions)
print("Precisão com rbf aumentando a precisão e ajustando o hiperparâmetro dos algoritmos usados: ", clf.score(features_test, labels_test))

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

```