Universidade de Brasília
Departamento de Estatística

Uma introdução ao método *Support Vector Machine*

Pedro Gabriel Moura

Projeto apresentado para o Departamento de Estatística da Universidade de Brasília como parte dos requisitos necessários para obtenção do grau de Bacharel em Estatística.

Brasília
2022

Pedro Gabriel Moura

Uma introdução ao método *Support Vector Machine*

Orientador(a): Prof. Dr. James Matos Sampaio

Projeto apresentado para o Departamento de Estatística da Universidade de Brasília como parte dos requisitos necessários para obtenção do grau de Bacharel em Estatística.

Brasília
2022

Agradecimentos

A primeira pessoa que preciso agradecer é meu pai, Rodolfo Moura, por não só ter tornado esse caminho possível de ser traçado como por ter ficado no meu pé ao longo dos anos para que eu não desistisse dos meus sonhos, sem você nada disso teria sido possível. Agradeço a minha família por todo o suporte e aos meus amigos, em especial Igor e Eduardo, que estiveram ao meu lado nos piores e melhores momentos, vocês me ajudaram a formar meu caráter e por isso sou eternamente grato. Alana, muito obrigado pelas incontáveis horas passadas me ouvindo, me motivando e por não ter me deixado desistir em nenhum momento, não sei o que seria de mim sem você ao meu lado.

Mãe, é nas vitórias que você faz mais falta, obrigado por tudo e saudades.

Resumo

A proposta deste Trabalho de Conclusão de Curso é criar um material em português introdutório ao método de Máquina de Vetores de Suporte para classificação (*Support Vector Machine Classifier*). O raciocínio para a criação das fórmulas utilizadas no modelo é explicado passo a passo e, em seguida, são apresentados a ideia e o raciocínio matemático por trás de seus hiperparâmetros. Após a apresentação do modelo, o mesmo é aplicado a três diferentes tipos de problema para solidificar seus conceitos e apresentar algumas de suas vantagens e desvantagens juntamente com seus resultados.

Palavras-chaves: *Support Vector Machine*, aprendizado de máquinas e modelo de classificação.

Abstract

The purpose of this Course Completion Work is to create a material in Portuguese introductory to the Support Vector Machine method for classification. The reasoning for the creation of the formulas used in the model is explained step by step and then the idea and mathematical reasoning behind its hyperparameters are presented. After the presentation of the model, it is applied to three different types of problem to solidify its concepts and present some of its advantages and disadvantages along with its results.

Key words: Support Vector Machine; Machine learning; Sorting model.

Lista de Tabelas

1 Tumor - Dados hipotéticos. 16
2 Tabela de contingência. 29
3 Imagem 1. 32
4 Imagem 2. 32
5 Imagem 3. 32
6 Imagem 4. 32
7 Imagens transformadas. 32
8 Frequência de dígitos. 34
9 Descrição das variáveis da base de reconhecimento de voz. 35
10 Exemplos de e-mails. 36
11 Frequência de palavras por e-mails. 36
12 Frequência de dígitos. 38
13 Bases treinos - *Text*. 42
14 *f1-score* - *Digits*. 43
15 Resumo dos modelos - *Text*. 45

Lista de Figuras

1 Definição de vetores. 11

2 Multiplicação entre vetor e escalar. 12

3 Somatório de vetores. 13

4 Projeção vetorial. 14

5 Exemplo de retas que segmentam o tipo de tumores. 17

6 *Widest road* na classificação de tumores. 18

7 Nova observação no problema de classificação de tumores. 18

8 Classificação de tumores - vetores $\vec{w}$ e $\vec{u}$. 19

9 Pontos de suporte na classificação do tipo de tumor. 21

10 Dados originais. 23

11 Dados após transformação. 23

12 Dados originais. 25

13 Linhas de decisão. 25

14 Modelo com erro de classificação. 25

15 Problema de classificação de tumor com classes sobrepostos. 26

16 Linha de decisão não sensível a outliers. 27

17 Definição de vetores. 28

18 Digito zero. 33

19 Digito um. 33

20 Digito dois. 33

21 Digito três. 33

22 Digito quatro. 33

23 Digito cinco. 33

24 Digito seis. 33

25 Digito sete. 33

26 Digito oito. 33

27 Digito nove. 33

28 Digito nove em formato de matriz. 33

29 Distribuição das variáveis por sexo. 35

30 Distribuição do tipo de e-mail. 37

31 Acurácia do modelo de classificação de imagem. 44

32 Acurácia do modelo de Reconhecimento de voz. 45

33 Acurácia do modelo Nº 1. 46

34 Acurácia do modelo Nº 2. 46

35 Acurácia do modelo Nº 3. 46

36 Acurácia do modelo Nº 4. 46

Sumário

Introdução 10

1 Referencial Teórico 11

1.1 **Definição de um Vetor** 11

1.1.1 Operações básicas 12

1.1.2 Produto escalar e projeção 13

1.2 **Definição de um Hiperplano** 14

1.3 **Método de Lagrange** 14

1.4 **Máquina de Vetores de Suporte para classificação** 15

1.4.1 Objetivo 15

1.4.2 Classificação de novos pontos 18

1.4.3 Largura da Rua 20

1.4.4 Casos não linearmente separáveis 23

1.4.5 Hiperparâmetro C 25

1.4.6 Função de perda 29

1.4.7 Validação cruzada de k-dobras (*k-fold cross-validation*) 30

1.4.8 Aplicação do modelo 30

2 Conjunto de dados 32

2.1 Classificação de imagens 32

2.2 Reconhecimento de voz 34

2.3 Classificação de texto 36

3 Metodologia 37

3.1 **Classificação de imagem** 37

3.1.1 Sanitização - MNIST 37

3.1.2 Análise exploratória - MNIST 38

3.1.3 Modelagem - MNIST 39

3.2 **Reconhecimento de Voz** . . . 39
3.2.1 Sanitização - Voz . . . 39
3.2.2 Análise exploratória - Voz . . . 40
3.2.3 Modelagem - Voz . . . 40
3.3 **Classificação de texto** . . . 40
3.3.1 Sanitização - Spam . . . 40
3.3.2 Análise exploratória - Spam . . . 41
3.3.3 Modelagem - Spam . . . 42
4 Resultados . . . 43
4.1 **Classificação de imagem** . . . 43
4.2 **Reconhecimento de voz** . . . 44
4.3 **Classificação de texto** . . . 44
Conclusão . . . 47
Referências . . . 48
Anexo . . . 50
.1 **Referencial teórico** . . . 51
.1.1 Imagens dos conceitos algébricos . . . 51
.1.2 Máquina de vetores de suporte para classificação . . . 53
.2 **Conjunto de dados** . . . 65
.2.1 Classificação de imagens . . . 65
.2.2 Reconhecimento de voz . . . 67
.2.3 Classificação de texto . . . 68
.3 **Metodologia** . . . 69
.3.1 Classificação de imagens . . . 70
.3.2 Reconhecimento de voz . . . 71
.3.3 Classificação de texto . . . 72
.4 **Resultados** . . . 77

Introdução

Situações em que a obtenção de dados é restrita são recorrentes ao longo da carreira do estatístico, seja pelo alto custo ou pela dificuldade de acesso. Por isso o conhecimento de técnicas que contornam esse problema é essencial. Uma delas é a Máquina de Vetores de Suporte, que, por utilizar apenas alguns pontos específicos para a criação da regra de decisão, pode ser considerado como um dos modelos fundamentais para profissionais das áreas de estatística e ciência de dados.

A proposta inicial do trabalho era apresentar o modelo de Máquina de Vetores de Suporte para classificação utilizando um problema de classificação da qualidade de vinhos. Esse problema, no entanto, requer métodos de tratamento de dados que são mais complexos que o próprio modelo, tornando-o inadequado como forma de apresentação. Além disso, também constatamos que somente um problema não seria suficiente para exemplificar a adaptabilidade do modelo. Dessa forma, optamos por apresentar o modelo e suas características utilizando três diferentes problemas, que são descritos a seguir.

Primeiro, o problema que popularizou a Máquina de Vetores de Suporte, reconhecimento de dígitos. Essencialmente, foi ensinado ao modelo diferenciar e classificar imagens de dígitos de zero a nove escritos à mão. A maior dificuldade nesse problema é garantir que o modelo aprenda a identificar quais pixels são essenciais para a diferenciação dos dígitos.

O segundo problema é o reconhecimento do gênero com base em medidas de posição e dispersão da gravação da voz de um indivíduo, tendo em vista a dificuldade em representar ondas sonoras de uma gravação de maneira tabular (uma das maiores dificuldades quando se trabalha com dados em formato de áudio). Vale ressaltar, no entanto, que não necessariamente todas as estatísticas geradas agregam informação significativa na diferenciação de gênero.

O último é um problema que está presente indiretamente no cotidiano de todos, classificação de e-mails entre “spam” e “não spam”. E-mails denominados “spam” são e-mails indesejados, em geral, propagandas. Apesar de parecer um problema mínimo, solucioná-lo é essencial para a experiência do usuário.

Ao final deste trabalho, o leitor será capaz de identificar, aplicar e analisar o resultado de problemas passíveis de serem resolvidos pela Máquina de Vetores de Suporte para classificação.

1 Referencial Teórico

A Máquina de Vetores de Suporte para classificação é um modelo de aprendizado supervisionado, no qual alguns conceitos algébricos, como vetores, hiperplanos, entre outros são a base para a compreensão do modelo. Tais conceitos são apresentados e revisados abaixo.

1.1 Definição de um Vetor

Vetores são segmentos de reta orientados que possuem três qualidades (FARIAS; KONZEN; SOUZA, 2020a): a **direção** que é determinada pela reta na qual o vetor se encontra, o **sentido**, que dita a orientação de atuação, e a **intensidade**, que também é designada como norma ou módulo. A intensidade de um vetor representa a sua grandeza e por isso o cálculo da intensidade de um vetor $\vec{x}$ é a distância euclidiana entre o ponto de origem e o final do vetor (FARIAS; KONZEN; SOUZA, 2020b):

$$||\vec{x}|| = \sqrt{x_1^2 + x_2^2 + \cdots + x_{n-1}^2 + x_n^2}$$

A variável n na equação acima é igual ao número de dimensões do espaço vetorial trabalhado.

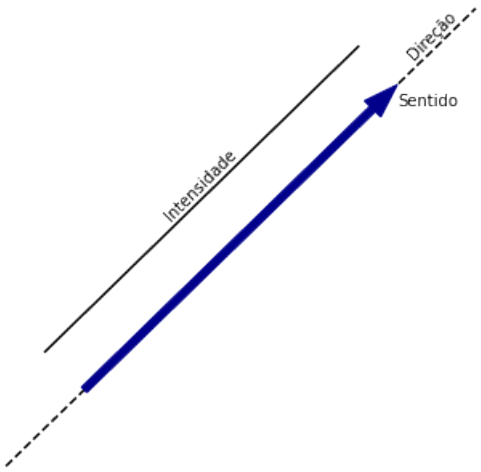


Figura 1: Definição de vetores.
Fonte: autoria própria.

1.1.1 Operações básicas

A **Multiplicação escalar** (BOLDRINI et al., 1986) de um vetor $\vec{x}$ por uma constante c aumenta a intensidade do vetor em c vezes:

$$\vec{x*} = c\vec{x} = cx_1 + cx_2 + \cdots + cx_{n-1} + cx_n$$

$$||\vec{x*}|| = \sqrt{c^2x_1^2 + c^2x_2^2 + \cdots + c^2x_{n-1}^2 + c^2x_n^2}$$

$$||\vec{x*}|| = \sqrt{c^2(2x_1^2 + 2x_2^2 + \cdots + x_{n-1}^2 + x_n^2)}$$

$$||\vec{x*}|| = c||\vec{x}||$$

Além da intensidade, o sinal de c altera também o sentido do vetor como demonstrado abaixo:

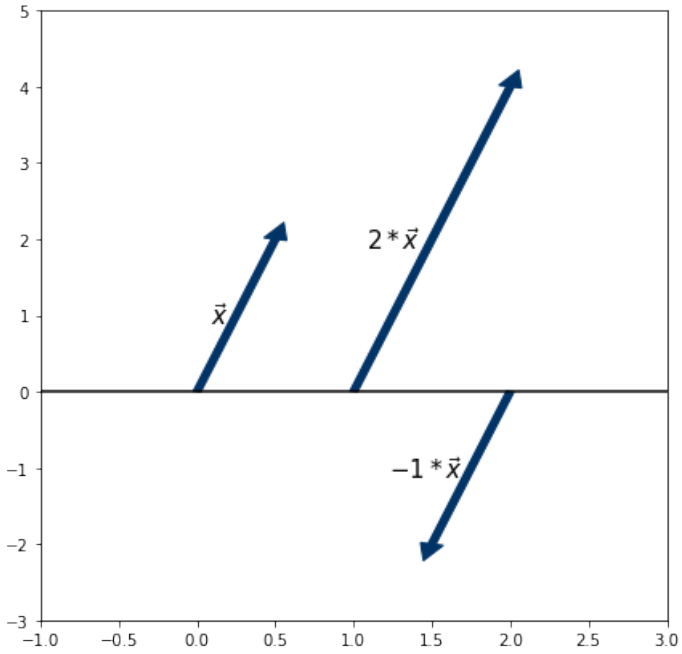


Figura 2: Multiplicação entre vetor e escalar.
Fonte: autoria própria.

A **Adição** e **Subtração** (BOLDRINI et al., 1986) de vetores é relativamente similar. A subtração é a soma de dois vetores, mas o segundo sendo multiplicado pelo valor -1. Suponha os vetores $\vec{v}$, $\vec{u}$ e $\vec{w} = \vec{v} + \vec{u}$ tem-se que:

$$\vec{w} = (v_1 + u_1, v_2 + u_2, \cdots, v_{n-1} + u_{n-1}, v_n + u_n)$$

$$w_p = v_p + u_p$$

Representação gráfica da adição de vetores:

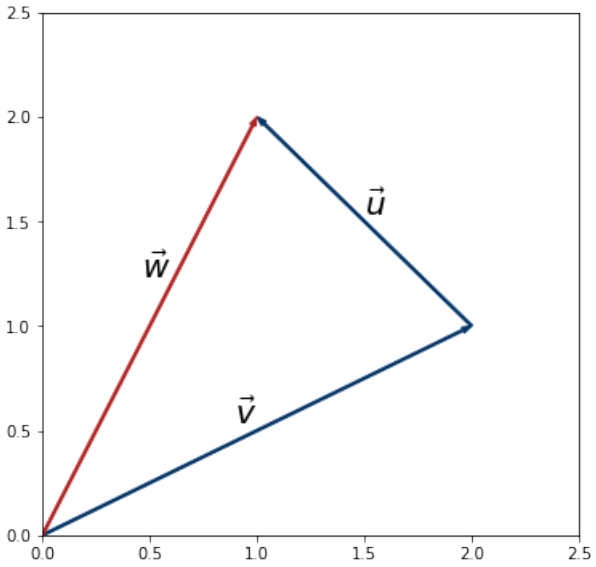


Figura 3: Somatório de vetores.
Fonte: autoria própria.

1.1.2 Produto escalar e projeção

O produto escalar, representado por $\vec{u} \cdot \vec{v}$, é definido por um valor real que pode ser calculado por $\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i$ (CALLIOLI; DOMINGUES; COSTA, 1990a). Supondo que θ é o ângulo entre os vetores, o produto escalar também é igual à:

$$\vec{u} \cdot \vec{v} = ||\vec{u}||\ ||\vec{v}||\ cos(\theta)$$

$$\vec{u} \cdot \frac{\vec{v}}{||\vec{v}||} = ||\vec{u}||cos(\theta)$$

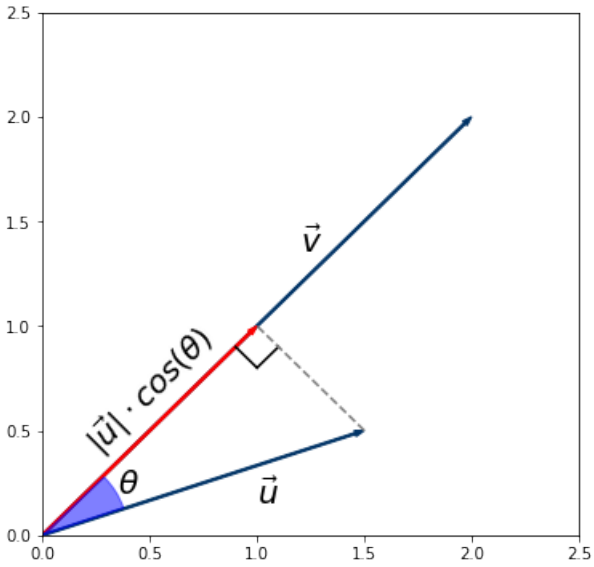


Figura 4: Projeção vetorial.
Fonte: autoria própria.

A parte à direita da igualdade da equação acima ($||\vec{u}||cos(\theta)$) representa a aplicação da intensidade do vetor $\vec{u}$ na reta que contém o vetor $\vec{v}$, portanto:

$$u_{proj\ v} = \vec{u} \cdot \frac{\vec{v}}{||\vec{v}||}$$

O valor $\vec{u} \cdot \frac{\vec{v}}{||\vec{v}||}$ é a representação escalar da projeção do vetor $\vec{u}$ sobre vetor $\vec{v}$, em outras palavras, o produto escalar acima representa a intensidade da projeção de $\vec{u}$ em $\vec{v}$.

1.2 Definição de um Hiperplano

Hiperplanos são figuras geométricas com $n-1$ dimensões (HASTIE; TIBSHIRANI; FRIEDMAN, 2008a) que dividem um espaço de n dimensões em subespaços, também, de n dimensões. No espaço tridimensional, os hiperplanos recebem o nome de planos; no bidimensional, reta, e no unidimensional, ponto. Sua forma algébrica em um espaço n dimensional, onde os valores a_i's e b são fixos, é igual a:

$$a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} + a_nx_n = b$$

1.3 Método de Lagrange

A derivada da função $f(x)$ representa a variação de $f(x)$. Um exemplo é a derivada da velocidade de um carro, que é igual à sua aceleração, portanto, quando a aceleração é

positiva a velocidade aumenta, quando negativa, a velocidade diminui e quando igual a zero, a velocidade se mantém constante. O método de Lagrange (CALLIOLI; DOMINGUES; COSTA, 1990b) para determinar os mínimos e máximos de uma função se baseia no fato de que todo mínimo e máximo, no seu ponto mais extremo, possui a derivada igual a zero.

$$L = f(x)$$
$$\frac{\partial L}{\partial x} = 0$$

O método acima é utilizado quando a função $f(x)$ não é limitada, para os casos limitados por funções $g_m(x) = 0$, onde m representa o número de funções restritivas, a função de Lagrange é igual à:

$$L(x, \lambda_i) = f(x) - \lambda_1 g_1(x) - \cdots - \lambda_m g_m(x)$$

$$\frac{\partial L(x, \lambda_i)}{\partial x} = 0$$
$$\frac{\partial L(x, \lambda_i)}{\partial \lambda_1} = 0$$
$$\cdots$$
$$\frac{\partial L(x, \lambda_i)}{\partial \lambda_m} = 0$$

A variável λ_i é conhecida como multiplicador de Lagrange. Existem diversas interpretações para essa variável, mas essencialmente ela é uma variável introduzida no problema para auxiliar na determinação dos pontos de mínimo e máximo, quando existente, dentro dos limites estabelecidos por $g_m(x) = 0$.

1.4 Máquina de Vetores de Suporte para classificação

A Máquina de Vetores de Suporte foi originalmente proposta por Vapnik e colaboradores em 1992 (BOSER; GUYON; VAPNIK, 1992), mas o algoritmo só se popularizou em 1994, devido ao sucesso em reconhecer dígitos escritos à mão.

1.4.1 Objetivo

O problema de diferenciação entre tumores malignos e benignos é um exemplo de uma das possíveis aplicações da Máquina de Vetores de Suporte para classificação. A título de exemplo, suponha que as características mais importantes para a definição do

tipo de tumor sejam o seu raio e textura e que foram amostrados quatro pacientes com tumor benigno e quatro com maligno, como indicado pela tabela abaixo:

ID	Raio	Textura	Tipo
1	0.3575	0.8103	Benigno
2	0.6508	0.5158	Benigno
3	0.4911	0.5348	Benigno
4	0.5904	0.4855	Benigno
5	0.9063	1.0384	Maligno
6	1.1018	0.5415	Maligno
7	0.9532	0.8217	Maligno
8	0.8495	1.0199	Maligno

Tabela 1: Tumor - Dados hipotéticos.
Fonte: autoria própria.

O objetivo deste método é encontrar um hiperplano, conhecido como hiperplano de decisão, que separe as observações amostrais de acordo com sua classificação. Para o caso acima, bidimensional, o algoritmo escolherá uma reta que separe tumores malignos de benignos. Com essa reta de decisão traçada, será possível classificar quaisquer novos tumores com base em sua localização no plano cartesiano em relação à reta de decisão. Para o exemplo apresentado, existem diversas retas que separam os dados de maneira satisfatória.

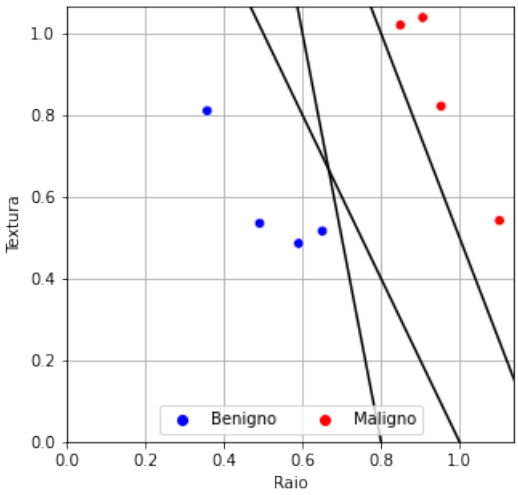


Figura 5: Exemplo de retas que segmentam o tipo de tumores.
Fonte: autoria própria.

A imagem acima apresenta o primeiro dos três grandes problemas direcionados ao objetivo descrito no início da sessão, sendo eles:

1. Existe um número muito grande de hiperplanos que separa satisfatoriamente os grupos. Sem uma definição clara, o algoritmo faria essa seleção aleatoriamente, impactando assim a classificação de novas observações;

2. Não são todos os problemas que possibilitam a separação linear das observações amostrais sem nenhum tipo de tratamento, o que faz com que o hiperplano não seja sempre uma solução possível;

3. O modelo não saberia como tratar problemas não separáveis, casos em que há sobreposição de observações de classes diferentes.

O primeiro grande problema é satisfatoriamente resolvido ao definir o conceito da "rua mais larga" (*widest road*) (WINSTON, 2010), que é a maior "rua" possível entre ambos os dados. As laterais da "rua" são conhecidas como hiperplanos de restrição, esses hiperplanos são paralelos e possuem a maior distância entre si possível. Entre os hiperplanos de restrição se encontra o hiperplano de decisão. Desta forma, só existe um hiperplano de decisão que separe linearmente as observações, quando possível, que satisfaz o objetivo do modelo e o conceito da "rua mais larga". O segundo e terceiro grandes problemas serão resolvidos nos próximos tópicos. Abaixo, é possível visualizar a "rua mais larga" no problema de classificação de tumores.

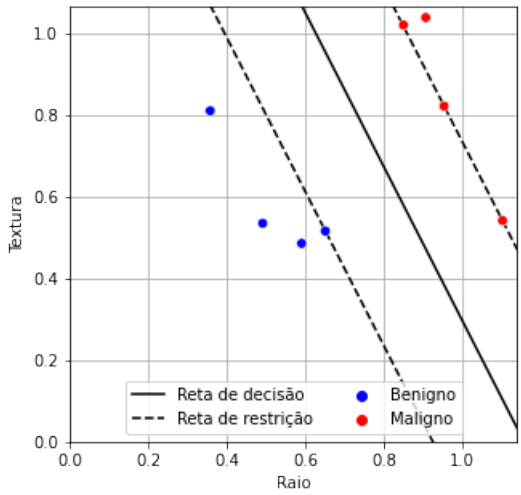


Figura 6: *Widest road* na classificação de tumores.
Fonte: autoria própria.

1.4.2 Classificação de novos pontos

Para exemplificar o raciocínio da classificação de um novo ponto u, foi utilizado o problema de classificação de tumores. Abaixo, segue a representação gráfica dos dados fornecidos anteriormente do novo ponto u e dos hiperplanos gerados.

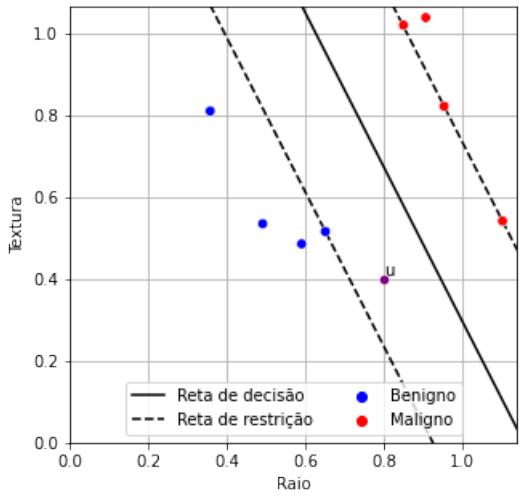


Figura 7: Nova observação no problema de classificação de tumores.
Fonte: autoria própria.

O ponto u no gráfico acima representa um novo paciente que ainda não sabe o tipo de tumor que possui (benigno ou maligno). É possível observar que esse paciente se encontra abaixo da reta de decisão, ou seja, no subespaço dos tumores benignos e, portanto, será classificado desta forma. Apesar de essa classificação ser nítida para o observador do gráfico, computacionalmente ela é realizada de outra forma. O primeiro

passo é definir o vetor $\vec{w}$ e $\vec{u}$: o primeiro é um vetor perpendicular à reta de decisão, possui intensidade desconhecida, porém fixa e parte da origem; o segundo se encontra entre a origem e o ponto u com sentido ao ponto u. O gráfico abaixo apresenta tais vetores:

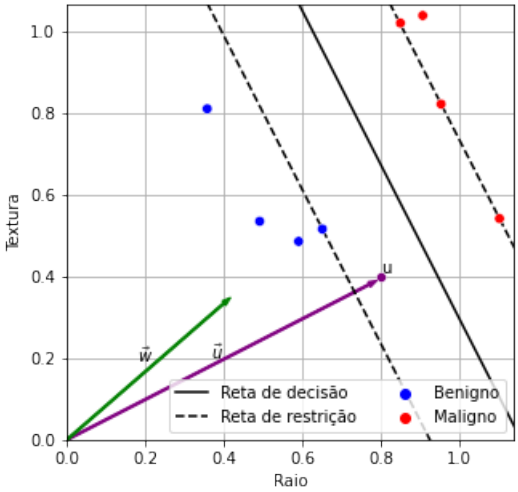


Figura 8: Classificação de tumores - vetores $\vec{w}$ e $\vec{u}$.
Fonte: autoria própria.

O produto escalar $\vec{u} \cdot \vec{w}$ resulta no comprimento da projeção vetorial de $\vec{u}$ sobre $\vec{w}$, portanto se a menor distância entre a origem e a reta de decisão, representado pela constante c, for menor que o comprimento da projeção vetorial de $\vec{u}$ sobre $\vec{w}$, o ponto estará no subespaço dos tumores malignos, caso contrário, se encontrará no subespaço dos tumores benignos.

$$\vec{u} \cdot \frac{\vec{w}}{||\vec{w}||} > c, \quad u \text{ é } maligno$$

$$\vec{u} \cdot \frac{\vec{w}}{||\vec{w}||} < c, \quad u \text{ é } benigno$$

A intensidade do vetor $\vec{w}$, como definida acima, é fixa, assim como o valor de c. Portanto, ao multiplicarmos ambos os lados das equações por $||\vec{w}||$ obtemos:

$$\vec{u} \cdot \vec{w} > -k, \quad u \text{ é } maligno$$

$$\vec{u} \cdot \vec{w} < -k, \quad u \text{ é } benigno$$

$$k = -c||\vec{w}||$$

A informação do tipo de tumor que o i-ésimo paciente possui, por conveniência matemática, foi representada numericamente pela variável y_i. Os tumores malignos foram

representados pelo valor $+1$ e benignos, por -1. Esta mudança possibilita condensar ambas as equações acima em apenas uma.

$$y_u = sinal[\vec{u} \cdot \vec{w} + k] \tag{1.4.1}$$

A variável y_u é igual a -1 quando $\vec{u} \cdot \vec{w} + k$ é negativo, e $+1$ caso contrário.

A equação acima não se restringe a um problema bidimensional, ela é conhecida como **regra de decisão** (HASTIE; TIBSHIRANI; FRIEDMAN, 2008b), uma das principais fórmulas utilizadas no modelo. Por enquanto, o vetor $\vec{w}$ segue desconhecido, porém, ciente de sua existência, é possível calcular a distância entre as retas de restrição.

1.4.3 Largura da Rua

Utilizando a mesma lógica aplicada à construção da equação (1.4.1) e o fato de que a constante b é igual à distância entre qualquer uma das retas de restrição à reta de decisão, é possível descrever todos os pacientes, pontos x_i, por meio das seguintes inequações:

$$\vec{x_i} \cdot \vec{w} + k - b \geq 0, \quad y_i = +1$$

$$\vec{x_i} \cdot \vec{w} + k + b \leq 0, \quad y_i = -1$$

Ainda na mesma lógica utilizada para encontrar equação (1.4.1), acrescentar a variavel y_i em ambas as inequações possibilita condensá-las na inequação abaixo.

$$y_i(\vec{x_i} \cdot \vec{w} + k) - b \geq 0$$

No primeiro parágrafo da introdução, foi mencionado que o modelo necessita de apenas alguns pontos específicos para a criação da regra de decisão, esses pontos são denominados pontos de suporte (HASTIE; TIBSHIRANI; FRIEDMAN, 2008b) e a partir deles são definidos os vetores de suporte que geram os hiperplanos de decisão e restrição. Os pontos de suporte são os pontos que se encontram nos hiperplanos de restrição, logo, os pontos descritos pela equação abaixo.

$$y_i(\vec{x_i} \cdot \vec{w} + k) - b = 0 \tag{1.4.2}$$

Aplicado ao problema de classificação do tipo de tumores, a equação (1.4.2) descreve os pontos destacados em amarelo no gráfico abaixo. Supondo os vetores $\vec{x_-}$ e $\vec{x_+}$ que vão da origem a um ponto da classe benigna, $y_i = -1$, e um da classe maligna, $y_i = +1$, respectivamente, e que obedecem a equação (1.4.2), é possível encontrar o vetor $(\vec{x_+} - \vec{x_-})$.

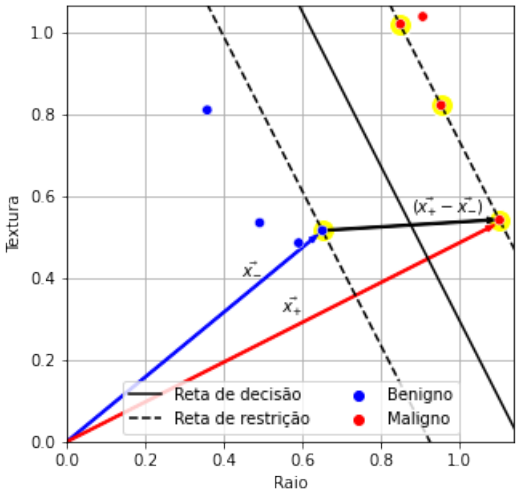


Figura 9: Pontos de suporte na classificação do tipo de tumor.
Fonte: autoria própria.

A distância d entre as retas de restrições é o produto escalar do vetor $(\vec{x_+} - \vec{x_-})$ e o vetor $\vec{w}$:

$$d = (\vec{x_+} - \vec{x_-}) \cdot \frac{\vec{w}}{||\vec{w}||}$$

Os vetores $\vec{x_+}$ e $\vec{x_-}$ podem ser descritos pela equação (1.4.2), resultando em:

$$d = \frac{((b-k) - (-b-k))}{||\vec{w}||}$$

$$d = \frac{2b}{||\vec{w}||} \tag{1.4.3}$$

De acordo com a definição apresentada da "rua mais larga", d é o maior valor possível e, portanto, a equação (1.4.3) passa a ser um problema de maximização. Maximizar $\frac{2b}{||\vec{w}||}$ é igual a maximizar $\frac{1}{||\vec{w}||}$, dado que b é constante, que é igual a minimizar $||\vec{w}||$, que igual a minimizar $\frac{||\vec{w}||^2}{2}$. Uma vez que a função $\frac{||\vec{w}||^2}{2}$ é uma função quadrática positiva, que é representada graficamente por uma parábola com a concavidade para cima, ela possui apenas um ponto onde sua derivada é igual a zero, o ponto de mínimo global e, portanto, ela será a função utilizada daqui em diante.

O problema de minimização é resolvido utilizando o método de Lagrange, onde m

representa o número de funções restritivas:

$$L = \frac{||\vec{w}||^2}{2} - \sum_{i}^{m} \lambda_i [y_i(\vec{w} \cdot \vec{x_i} + k) - b] \tag{1.4.4}$$

Na equação acima, tem-se que $\frac{||\vec{w}||^2}{2}$ é a função a ser minimizada, $y_i(\vec{w} \cdot \vec{x_i} + k) - b$ a função de restrição determinada pela equação (1.4.2) e λ_i o multiplicador de Lagrange i. Calculando as derivadas, obtemos:

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_{i}^{m} \lambda_i y_i \vec{x_i} = 0$$

$$\vec{w} = \sum_{i}^{m} \lambda_i y_i \vec{x_i} \tag{1.4.5}$$

$$\frac{\partial L}{\partial k} = \sum_{i}^{m} \lambda_i y_i k = 0$$

$$\sum_{i}^{m} \lambda_i y_i k = 0 \tag{1.4.6}$$

Substituindo as equações (1.4.6) e (1.4.5) em (1.4.4) tem-se:

$$L = b \sum_{i}^{m} \lambda_i - \frac{1}{2} (\sum_{i}^{m} \sum_{j}^{m} \lambda_i \lambda_j y_i y_j \vec{x_i} \cdot \vec{x_j}) \tag{1.4.7}$$

Ao analisarmos a equação (1.4.7), observamos que b é a distância entre as retas de restrição e a reta de decisão, logo um valor fixo, λ_i o multiplicador de Lagrange i, y_i e y_j que dependem da classe de x_i e x_j, respectivamente, e o produto escalar $\vec{x_i} \cdot \vec{x_j}$. Portanto o valor máximo de d depende única e exclusivamente dos produtos escalares $\vec{x_i} \cdot \vec{x_j}$ e, por isso, o problema de maximização passa a ser um problema de otimização dos pontos amostrais.

Durante o processo de definir qual a largura da "rua"foi identificada a equação (1.4.5) que determina o vetor $\vec{w}$. Esta equação permite que o hiperplano de decisão, equação (1.4.1), seja escrito também como um problema de otimização.

$$y_u = sinal[\sum_{i}^{m} \lambda_i y_i \vec{u} \cdot \vec{x_i} + k] \tag{1.4.8}$$

Após a solução do problema de otimização, determinar o valor de b, k e o hiperplano

de decisão passa a ser trivial.

1.4.4 Casos não linearmente separáveis

O segundo grande problema, sobre os dados não serem linearmente separáveis, é resolvido com um método chamado truque de kernel (*kernel trick*) (HASTIE; TIBSHIRANI; FRIEDMAN, 2008c).Tal método consiste em aplicar uma série de transformações matemáticas, $h_i(\vec{x_i})$, para aumentar a quantidade de dimensões no intuito de que o problema seja linearmente separável em uma das novas dimensões criadas. Exemplo:

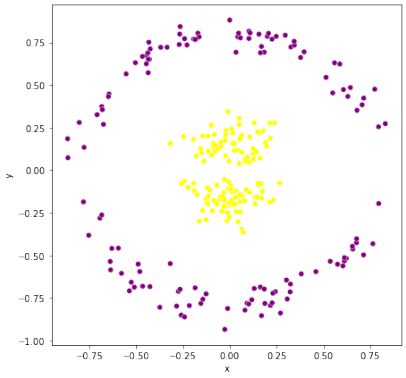


Figura 10: Dados originais.
Fonte: autoria própria.

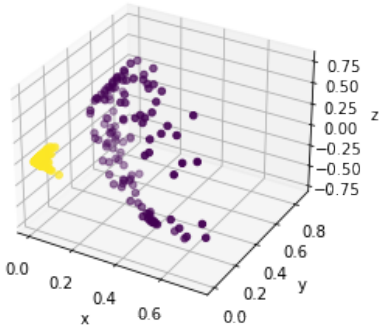


Figura 11: Dados após transformação.
Fonte: autoria própria.

Como é possível visualizar acima, os dados que originaram a Figura 10 são separáveis, porém não linearmente. Após a transformação, fica claro, na Figura 11, que os dados passam a ser linearmente separáveis. Como o aumento de dimensõesé aplicado nos vetores, a complexidade do modelo não aumenta e os problemas de otimização passam a ser:

$$L = b\sum_{i}^{m}\lambda_i - \frac{1}{2}(\sum_{i}^{m}\sum_{j}^{m}\lambda_i\lambda_j y_i y_j h(\vec{x_i})\cdot h(\vec{x_j})) \tag{1.4.9}$$

$$y_u = sinal[\sum_{i}^{m}\lambda_i y_i h(\vec{u})\cdot h(\vec{x_i}) + k] \tag{1.4.10}$$

O produto vetorial $h_i(\vec{x_i})\cdot h_i(\vec{x'}_i)$ é igual à $h_1(x_1)h_1(x'_1) + h_2(x_2)h_2(x'_2) + \cdots + h_n(x_n)h_n(x'_n)$ em que n representa o número de dimensões após a aplicação truque de kernel, tal produto pode ser representado pela função $K(x, x')$. Abaixo seguem alguns kerneis famosos na literatura:

- $K(\vec{x}, \vec{x'}) = (\vec{x}\cdot\vec{x'} + 1)^p$ - Kernel Polinomial;

- $K(\vec{x}, \vec{x'}) = e^{\gamma(\vec{x}-\vec{x'})^2}$ - Kernel RBF;
- $K(\vec{x}, \vec{x'}) = tanh(\eta\vec{x} \cdot \vec{x'} + \upsilon)$ - Kernel Sigmoid.

Exemplo do truque de kernel retirado de (HASTIE; TIBSHIRANI; FRIEDMAN, 2008c) página 424: suponha um problema bidimensional e que foi selecionado o Kernel Polinomial com $p = 2$ para a transformação resultando em:

$$K(\vec{x}, \vec{x'}) = (\vec{x} \cdot \vec{x'} + 1)^2$$

$$K(\vec{x}, \vec{x'}) = (x_1x'_1 + x_2x'_2 + 1)^2$$

$$K(\vec{x}, \vec{x'}) = x_1x'_1 + x_1^2x_1'^2 + 2x_1x'_1x_2x'_2 + x_2^2x_2'^2 + x_2x'_2 + 1$$

Os dados originais que possuíam duas dimensões, mas, após o truque de kernel, passam a ter seis dimensões:

$$h_1(\vec{x}) = x_1, \quad h_2(\vec{x}) = x_1^2, \quad h_3(\vec{x}) = \sqrt{2}x_1x_2$$

$$h_4(\vec{x}) = x_2^2, \quad h_5(\vec{x}) = x_2 \; e \; h_6(\vec{x}) = 1$$

Apesar do truque de kernel resolver o problema de dados não linearmente separáveis, porém separáveis, ele gera dois novos problemas:

1. Os kerneis aumentam a dimensionalidade dos dados proporcionalmente à dimensionalidade original e, além disso, o aumento costuma ser considerável, muitas vezes mais que dobrando o número de dimensões. Essa característica do truque de kernel tende a aumentar o custo computacional, visto que o número de subconjuntos de dimensões aumenta demasiadamente e apenas um desses subconjuntos será útil para a separação dos dados (HASTIE; TIBSHIRANI; FRIEDMAN, 2008d).
2. O número de dimensões aumenta até que seja possível classificar corretamente todas as observações amostrais, porém, quando o problema não é separável, o número de dimensões pode tender ao infinito. Ainda que o modelo consiga encontrar um subconjunto de dimensões em que seja possível separar os dados corretamente, esse conjunto seria tão grande e tão específico para base treino que o modelo perderia sua capacidade de generalizacão (HASTIE; TIBSHIRANI; FRIEDMAN, 2008c).

Abaixo, segue um exemplo do segundo problema criado pelo truque de kernel. A primeira figura representa os dados originais em que existe a sobreposição de classes e

a segunda figura representa as linhas de decisão criadas em dimensões além dos dados originais e adaptada para duas dimensões:

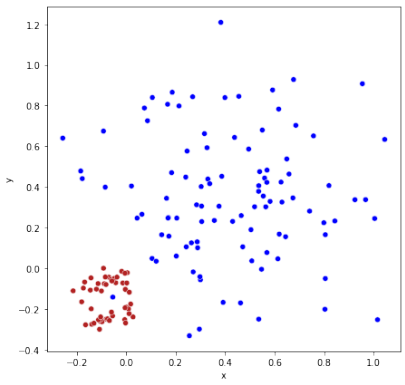


Figura 12: Dados originais.
Fonte: autoria própria.

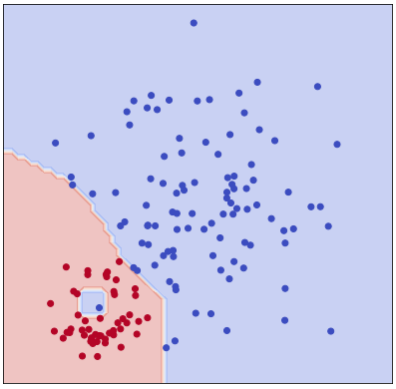


Figura 13: Linhas de decisão.
Fonte: autoria própria.

Quanto aos problemas originados pelo truque de kernel, o aumento do custo computacional gerado pelo método não possui solução, porém a solução do segundo problema ameniza esse aumento. O problema de perda de generalização é resolvido ao adaptar o modelo para que passe a ser aceitável um certo número de classificações incorretas, desta forma, o modelo mantém seu poder de generalizacão, como no caso abaixo:

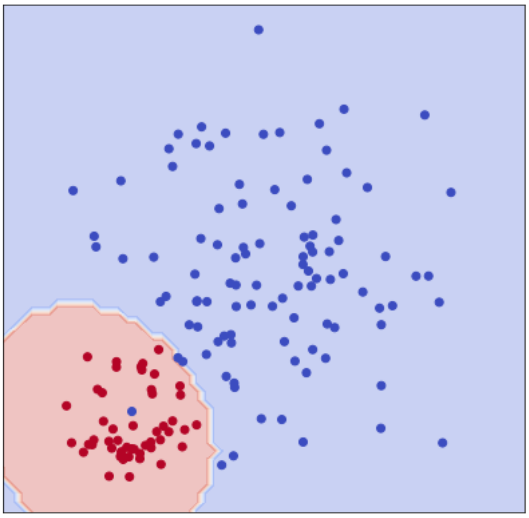


Figura 14: Modelo com erro de classificação.
Fonte: autoria própria.

1.4.5 Hiperparâmetro C

A capacidade de um modelo de classificar erroneamente observações de maneira controlada é essencial, apesar de contraintuitivo. Modelos de aprendizado de máquina são

necessários em situações em que os dados contêm uma certa aleatoriedade, o que significa que é possível possuir duas observações com os mesmos *inputs* e de classes diferentes. Essa aleatoriedade intrínseca dos problemas torna o muitas vezes impossível de prever a classificação de todos os objetos corretamente e, portanto, uma taxa de erro sempre existirá. Um modelo que se adapta completamente à sua base de treino muitas vezes se torna muito específico, o que faz com que sua taxa de erro aumente muito quando aplicado ao restante da população. Suponha que no problema de tumor anterior houvesse dados sobrepostos desta maneira:

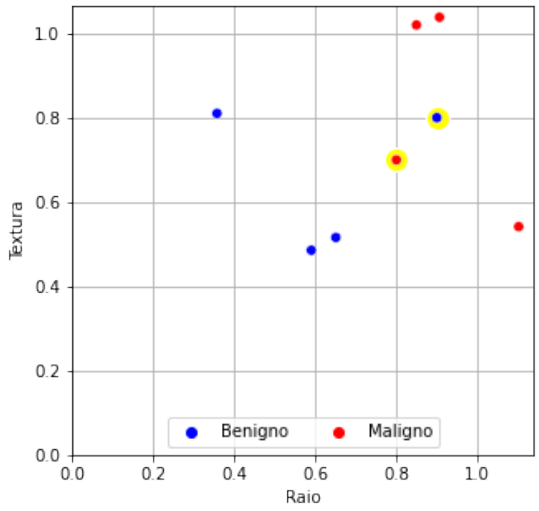


Figura 15: Problema de classificação de tumor com classes sobrepostos.
Fonte: autoria própria.

Os pontos destacados em amarelo do gráfico acima são os pontos com *inputs* alterados em relação ao problema original. Devido a esses pontos, o problema passa a ser não separável. No caso acima, o desejado é que o modelo ignore os pontos destacados em amarelo e mantenha os hiperplanos anteriormente criados, como abaixo:

A equação anteriormente utilizada para descrever os pontos amostrais, $y_i(\vec{x_i} \cdot \vec{w} + k) \geq b$, consegue descrever os pontos que não se encontram entre as retas de restrição, porém falha ao descrever os pontos internos. Existem duas alterações na fórmula original que fazem com que ela passe a descrever todos os pontos (HASTIE; TIBSHIRANI; FRIEDMAN, 2008e), internos e externos, sendo elas:

$$y_i(\vec{x_i} \cdot \vec{w} + k) \geq b - \varepsilon_i \tag{1.4.11}$$

$$y_i(\vec{x_i} \cdot \vec{w} + k) \geq b(1 - \varepsilon_i) \tag{1.4.12}$$

A equação (1.4.11) é a maneira mais intuitiva de adaptar a equação original, visto

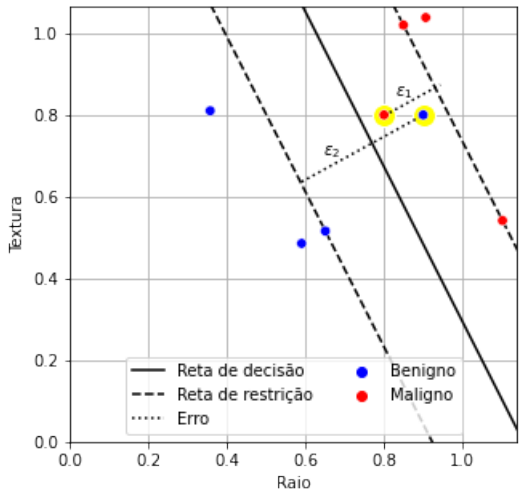


Figura 16: Linha de decisão não sensível a outliers.
Fonte: autoria própria.

que, neste caso, ε_i é igual a zero para os pontos externos aos hiperplanos de restrição e igual à distância entre o ponto e o hiperplano de restrição para os pontos internos. Porém, ao adaptar o modelo para equação (1.4.11), o problema de otimização não converge (HASTIE; TIBSHIRANI; FRIEDMAN, 2008e) e, por isso, apesar de ser a forma mais intuitiva, a equação (1.4.11) não é utilizada.

O problema de otimização, quando realizado com a equação (1.4.12), converge. Nesta equação, a variavel ε_i é a distância relativa do ponto ao hiperplano de restrição, logo, todo ε_i é maior ou igual a zero e a classificação incorreta somente ocorre quando $\varepsilon_i > 1$, que é quando o ponto ultrapassa o hiperplano de decisão (HASTIE; TIBSHIRANI; FRIEDMAN, 2008e). Os valores de ε_i são atrelados a uma constante K pela equação $\sum \varepsilon_i \leq K$, desta forma, o máximo de classificações incorretas é limitada sempre ao valor de K (HASTIE; TIBSHIRANI; FRIEDMAN, 2008e).

Utilizando a equação (1.4.12), a função a ser minimizada se torna $\frac{||\vec{w}||^2}{2} + C\sum \varepsilon_i$, em que C é o hiperparâmetro que penaliza o número de erros. Intuitivamente, quanto maior o valor do hiperparâmetro C, maior a punição por erro e, portanto, menos tolerante o modelo é à classificações incorretas, fazendo com que as margens da "rua" tendam a se adaptar mais a *outliers*.

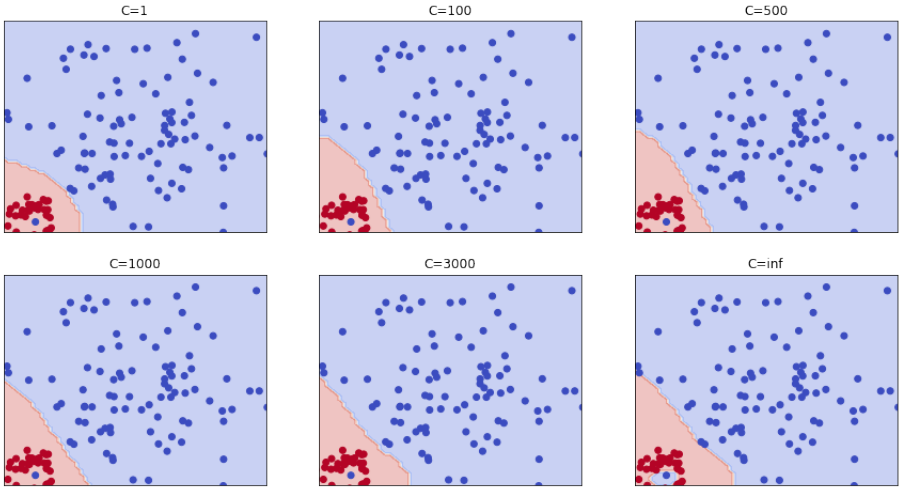


Figura 17: Definição de vetores.
Fonte: autoria própria.

O valor de C determina o quão aceitável são pontos classificados incorretamente e dentro da "rua". Um efeito secundário é na largura da "rua"e, consequentemente, sua posição, pois quanto mais pontos são aceitos ou classificados incorretamente, mais larga tende ser a "rua" sem manter necessariamente sua posição central. Na Figura 17, é possível observar que o hiperplano de decisão muda de formato conforme C varia.

O novo problema de minimização (HASTIE; TIBSHIRANI; FRIEDMAN, 2008e) resulta em:

$$L = \frac{1}{2}||\vec{w}||^2 + C\sum_{i=1}^{m}\varepsilon_i - \sum_{i=1}^{m}\lambda_i[y_i(\vec{x_i}\cdot\vec{w}+k) - b(1-\varepsilon_i)] - \sum_{i=1}^{m}\mu_i\varepsilon_i \tag{1.4.13}$$

Derivadas:

$$\vec{w} = \sum_{i=1}^{m}\lambda_i y_i x_i$$

$$0 = \sum_{i=1}^{m}\lambda_i y_i$$

$$\lambda_i = C - \mu_i$$

Substituindo as derivadas em (1.4.13):

$$L = b\sum_{i}^{m}\lambda_i - \frac{1}{2}(\sum_{i}^{m}\sum_{j}^{m}\lambda_i\lambda_j y_i y_j \vec{x_i}\cdot\vec{x_j}) \tag{1.4.14}$$

$$0 \leq \lambda_i \leq C \tag{1.4.15}$$

Com a definição da "rua mais larga", do hiperparâmetro C e do truque de kernel, o modelo consegue lidar com dados sobrepostos e com uma solução não linear. Resta somente definir como ajustar os hiperparâmetros do modelo para que este tenha a melhor performance possível.

1.4.6 Função de perda

Conseguir quantificar o quão bom é um modelo é um dos passos mais importantes durante a modelagem, não necessariamente a taxa de acerto é a melhor medida para determinar a performance. Por exemplo, suponha que o grupo "A" compõe 90% da população, se o modelo inferir que, independente dos *inputs*, uma nova observação sempre pertence ao grupo "A", o modelo obterá uma taxa de acerto aproximada de 90%, mesmo não tendo aprendido nada sobre o comportamento dos dados. A natureza do problema dita qual a melhor função de perda, algumas funções (SASAKI, 2007) comumente utilizadas para classificação são (os valores de VP, VN, FP e FN são explicados logo após a apresentação das funções):

- Acurácia: $\frac{VP+VN}{VP+VN+FP+FN}$
- F1: $\frac{VP}{VP+\frac{1}{2}(FP+FN)}$

		Class. correta	
		Verdadeiro	Falso
Class. modelo	Verdadeiro	VP	FP
	Falso	FN	VN

Tabela 2: Tabela de contingência.
Fonte: autoria própria.

O valor de VP é igual ao número de pontos classificados corretamente como pertencentes ao grupo 1, VN o número de pontos classificados corretamente como pertencentes ao grupo 2, FP a quantidade de pontos classificados erroneamente como pertencentes ao grupo 1 e FN, erroneamente ao grupo 2.

1.4.7 Validação cruzada de k-dobras (*k-fold cross-validation*)

Dados provenientes de problemas passiveis de serem resolvidos por aprendizado de máquina tendem a possuir certo ruído. É chamado de ruído o conjunto de dados com comportamento inesperados. Dessa forma, um modelo que se adapta também ao ruído durante o treino perde seu poder de generalização para novas informações. Esse fenômeno é conhecido como *overfitting*.

A preocupação com o *overfitting* é muito comum durante a seleção dos hiperparâmetros de um modelo, no intuito de medi-lo, é comum dividir aleatoriamente a amostra entre base teste e treino, sendo que normalmente a base treino é muito maior que a base teste (para que o modelo não saia prejudicado por falta de informações durante o aprendizado). Dessa forma, o modelo aprende somente com base nas informações da base de treino e depois é aplicada a função perda em suas previsões para base teste e seu real valor. A solução acima gera um novo problema, ao realizar a divisão aleatória existe a possibilidade de todos os valores considerados atípicos irem para base teste, isso fará com que o modelo seja classificado como ruim injustamente. Uma solução para o problema acima é a validação cruzada de k-dobras (*k-fold cross-validation*) (BOWNE-ANDERSON, 2017). O método segrega a amostra em k partes iguais, para explicação assuma $k = 3$ (parte 1, parte 2 e parte 3), e com as partes 1 e 2 o modelo terá seu aprendizado e testado na parte 3, depois um novo modelo com os mesmos hiperparâmetros tem seu aprendizado com base nas partes 1 e 3 e testado na parte 2 e, por último, um novo modelo com os mesmo hiperparâmetros e com seu aprendizado realizado com base nas parte 2 e 3 e testado na parte 1. Os modelos são descartados e suas métricas são sumarizadas, dessa forma, é possível obter uma estimativa mais confiável do quão bom os hiperparâmetros selecionados são.

1.4.8 Aplicação do modelo

Tendo em mente seu funcionamento, bem como suas vantagens e desvantagens, é possível identificar as situações de aplicação.

Devido ao truque de kernel, o custo computacional cresce proporcionalmente com o aumento do número de dimensões M. Já quanto ao número de linhas N, o método utiliza de todos os pontos amostrais para identificar quem são os pontos de suporte e criar as retas de decisão e restrições, logo quando maior N maior o custo computacional. Portanto, quanto maior for a matrizes (NxM) menos indicado é o modelo.

Em contrapartida, por utilizar o hiperplano que maximiza a distância entre observações de classes diferentes como hiperplano de decisão o modelo tende a funcionar bem para problemas com poucas observações. Além disso o modelo necessita de poucos hiperparâmetros, sendo assim, um modelo fácil e rápido de ser aplicado.

2 Conjunto de dados

Os bancos de dados utilizados para a análise foram retirados do site *https://www.kaggle.com/*, que é uma plataforma criada para o compartilhamento de base de dados, modelos, desafios e outros temas relativo à ciência e análise de dados. As três bases retiradas são contextualizadas e explicadas nas subseções abaixo.

2.1 Classificação de imagens

O primeiro banco de dados foi postado com o nome "MNIST Digit Recognizer"(ANIMATRONBOT, 2018). Ele foi construído com base no banco de imagens MNIST (*Modified National Institute of Standards and Technology database*).

MNIST é uma base criada com a foto dos dígitos de 0 a 9 escritos por alunos do ensino médio e empregados da empresa estadunidense "*Census Bureau*"(LECUN; CORTES; BURGES,). As imagens possuem a resolução de 28x28 pixels com o fundo da imagem preto e os dígitos em tons de branco.

A base utilizada contém uma amostra das imagens do MNIST, já tratadas. Nesse caso, o tratamento das imagens constitui-se em numerar os pixels e trazer cada pixel como uma coluna, como no exemplo abaixo:

Pixel			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabela 3: Imagem 1.
Fonte: autoria própria.

Pixel			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabela 4: Imagem 2.
Fonte: autoria própria.

Pixel			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabela 5: Imagem 3.
Fonte: autoria própria.

Pixel			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabela 6: Imagem 4.
Fonte: autoria própria.

Image	coluna 1	coluna 2	...	coluna 15	coluna 16
Image 1	Pixel 0	Pixel 1	...	Pixel 14	Pixel 15
Image 2	Pixel 0	Pixel 1	...	Pixel 14	Pixel 15
Image 3	Pixel 0	Pixel 1	...	Pixel 14	Pixel 15
Image 4	Pixel 0	Pixel 1	...	Pixel 14	Pixel 15

Tabela 7: Imagens transformadas.
Fonte: autoria própria.

Como as imagens no problema em questão são de 28x28 pixel, são criadas 784

colunas, onde cada célula contém valores de 0 a 255. Quanto menor o valor da célula, mais próximo a cor de preto e quanto maior, mais perto da cor branca. Sabendo a resolução da imagem, é possível reconstruí-la, abaixo seguem algumas das imagens reconstruídas a partir da base.

Figura 18: Digito zero.
Fonte: autoria própria.

Figura 19: Digito um.
Fonte: autoria própria.

Figura 20: Digito dois.
Fonte: autoria própria.

Figura 21: Digito três.
Fonte: autoria própria.

Figura 22: Digito quatro.
Fonte: autoria própria.

Figura 23: Digito cinco.
Fonte: autoria própria.

Figura 24: Digito seis.
Fonte: autoria própria.

Figura 25: Digito sete.
Fonte: autoria própria.

Figura 26: Digito oito.
Fonte: autoria própria.

Figura 27: Digito nove.
Fonte: autoria própria.

Considerando que cada pixel recebe um número de 0 a 255, é possível reescrever as imagens como uma matriz, abaixo tem-se a Figura 26 em formato de matriz:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	43	7	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	29	85	85	129	192	191	222	253	201	174	85	43	0	0	0	0	0	0
9	0	0	0	0	0	0	0	6	66	145	240	253	253	253	254	253	253	253	253	254	253	234	52	0	0	0	0	0
10	0	0	0	0	0	22	73	191	253	254	253	248	190	102	85	84	84	84	84	85	120	222	232	28	0	0	0	0
11	0	0	0	0	0	129	253	253	165	130	42	38	0	0	0	0	0	0	0	0	0	39	253	42	0	0	0	0
12	0	0	0	0	105	254	224	80	0	0	0	0	0	0	0	0	0	0	0	0	0	22	254	210	0	0	0	0
13	0	0	0	0	148	253	47	0	0	0	0	0	0	0	0	0	0	0	0	0	29	188	253	253	0	0	0	0
14	0	0	0	0	246	78	2	0	0	0	0	0	0	0	0	0	0	9	190	240	253	221	38	0	0	0	0	0
15	0	0	0	0	254	111	6	0	0	0	0	0	0	0	0	0	17	187	255	253	213	63	0	0	0	0	0	0
16	0	0	0	0	104	183	162	80	0	0	0	0	0	0	45	0	9	187	253	254	182	21	0	0	0	0	0	0
17	0	0	0	0	0	0	107	212	212	89	0	13	148	114	123	192	210	254	254	151	27	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	6	63	11	63	101	253	253	182	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	31	214	252	161	21	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27	175	253	179	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	54	229	254	138	4	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	31	219	254	195	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	163	253	253	71	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	88	252	248	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	100	233	239	63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	25	42	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 28: Digito nove em formato de matriz.
Fonte: autoria própria.

A base em questão contém 42 mil fotos e 785 colunas, os 784 pixels e uma coluna indicando qual o dígito contido na imagem. Abaixo segue a quantidade de imagens por dígito:

Digito	Frequência
0	4132
1	4684
2	4177
3	4351
4	4072
5	3795
6	4137
7	4401
8	4063
9	4188

Tabela 8: Frequência de dígitos.
Fonte: autoria própria.

2.2 Reconhecimento de voz

A base contém estatísticas acústicas de 3.168 áudios, sendo 2 com *inputs* duplicados e, portanto, removidos. Metade da base foi construída com base na voz de pessoas do gênero feminino e a outra metade, do gênero masculino. Os dados foram postados pela usuária "KORY BECKER"com o nome "*Gender Recognition by Voice*"(BECKER, 2016).

Os áudios dentro da frequência sonora 0hz-280hz, *range* da voz humana, foram pré-tratados pelo software estatístico R com os pacotes "seewave" e "tuneR". Abaixo as distribuições das colunas da base e suas descrições.

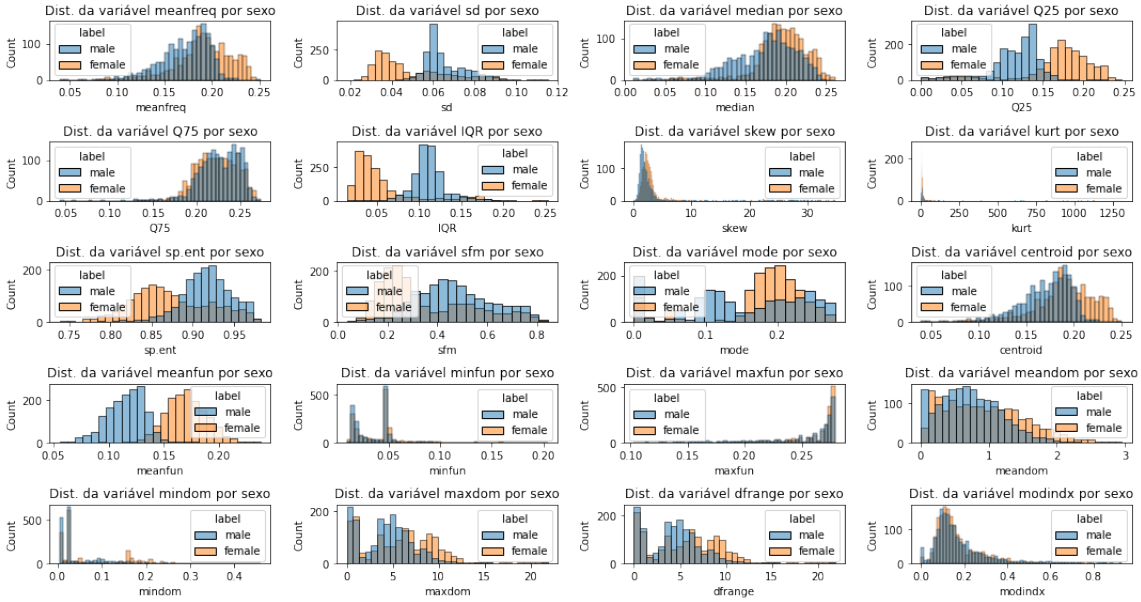


Figura 29: Distribuição das variáveis por sexo.
Fonte: autoria própria.

Coluna	**Descrição**
meanfreq	Frequência média em kHz
sd	Desvio padrão da frequência
median	Mediana da frequência em kHz
Q25	Primeiro quartil da frequência em kHz
Q75	Terceiro quartil da frequência em kHz
IQR	Intervalo interquartilico da frequência em kHz
skew	Skewness da frequência
kurt	Kurtosis da frequência
sp.ent	Entropia espectral do sinal
sfm	Nivelamento espectral
mode	Moda da frequência em kHz
centroid	Centroide da frequência
peakf	Pico da frequência
meanfun	Média da frequência fundamental medida através do sinal acústico
minfun	Frequência fundamental mínima medida através do sinal acústico
maxfun	Frequência fundamental máxima medida através do sinal acústico
meandom	Média da frequência dominante medida através do sinal acústico
mindom	Mínimo da frequência dominante medido através do sinal acústico
dfrange	Faixa de frequência dominante medida através do sinal acústico
modindx	Índice de modulação. Calculado como a diferença absoluta acumulada entre medições adjacentes de frequências fundamentais dividida pela faixa de frequência
label	Sexo do emissor do som

Tabela 9: Descrição das variáveis da base de reconhecimento de voz.
Fonte: autoria própria.

2.3 Classificação de texto

A última base (BISWAS, 2020) contém informações de 5.172 e-mails em inglês, sendo 541 e-mails duplicados e, por isso, foram removidos. O texto de cada e-mail teve seus caracteres numéricos e especiais, com exceção do "espaço", removidos e todos os caracteres maiúsculos transformados em minúsculos. O conjunto de palavras possíveis dentro dos e-mails, *substrings* entre o caracter especial espaço, foi transformado no conjunto de variáveis da base de dados. Cada linha representa um e-mail e cada célula da base representa a frequência da palavra (nome da coluna) dentro do e-mail. Exemplo de construção:

E-mails	**Texto**
E-mail 1	Exemplo de e-mail exemplo
E-mail 2	Um e-mail modelo
E-mail 3	E-mail base
E-mail 4	Base de um modelo

Tabela 10: Exemplos de e-mails.
Fonte: autoria própria.

E-mails	**exemplo**	**de**	**e-mail**	**um**	**modelo**	**base**
E-mail 1	2	1	1	0	0	0
E-mail 2	0	0	1	1	1	0
E-mail 3	0	0	1	0	0	1
E-mail 4	0	1	0	1	1	1

Tabela 11: Frequência de palavras por e-mails.
Fonte: autoria própria.

Após o tratamento e a criação da tabela, foram removidas colunas de tal forma que restassem apenas as 3.000 palavras mais comuns em todos e-mails e as colunas que demarcam o índice do e-mail e a classe do e-mail (spam e não spam). Abaixo, a concentração do tipo de e-mails:

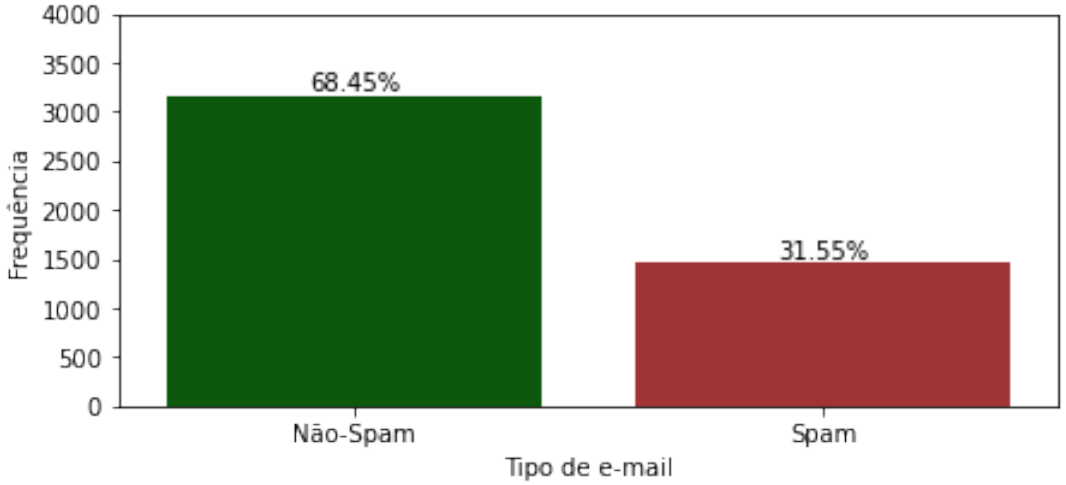


Figura 30: Distribuição do tipo de e-mail.
Fonte: autoria própria.

3 Metodologia

O modelo de classificação foi aplicado em três diferentes problemas: classificação de imagens, reconhecimento de voz e classificação de texto.

A metodologia para aplicação do modelo, em todos os problemas, foi dividida em três fases. A primeira fase é a de "sanitização", em que ocorreu a compreensão e a validação dos dados. Na fase seguinte, "análise exploratória", que consiste na investigação das relações entre variáveis, ocorreu a criação de estratégias de modelagem. Na terceira fase, "Modelagem", foram definidos quais os melhores hiperparâmetros para o problema em questão.

3.1 Classificação de imagem

Esse problema consiste no reconhecimento de dígitos escritos à mão da amostra da base MNIST *database* (*Modified National Institute of Standards and Technology database*).

3.1.1 Sanitização - MNIST

A base é não balanceada, porém a diferença entre rótulos não é significativa o suficiente para exigir algum método de correção como *undersampling* e não há nenhuma imagem repetida. Não há valores faltantes.

Classe	Frequência
0	4132
1	4684
2	4177
3	4351
4	4072
5	3795
6	4137
7	4401
8	4063
9	4188

Tabela 12: Frequência de dígitos.
Fonte: autoria própria.

Apesar de não possuir valores faltantes, cerca de 80% da base é constituída por zeros. Porém os zeros não representam a falta de quantidade e, sim, a cor preto.

3.1.2 Análise exploratória - MNIST

Cada imagem é constituída por 784 pixels (28x28), o que significa que cada imagem possui 784 dimensões. Devido ao truque de kernel aumentar a dimensão dos dados proporcionalmente, o custo computacioanl passa a ser muito grande. Para diminuir esse custo, foi aplicado o método de redução de dimensionalidade PCA (*Principal Component Analysis*).

O PCA (JOLLIFFE, 2017) é um método de redução de dimensionalidade algébrico para variáveis numéricas que utiliza transformações ortogonais para definir os componentes principais. Com os componentes principais, é possível criar um banco com igual ou menos variáveis que o original, sendo que quanto menos variáveis, menor o poder do banco criado de explicar a variabilidade do banco original. Utilizando o método PCA, foi criada uma base com 87 dimensões que explica cerca de 90% da variabilidade de base original.

Após a redução da dimensionalidade, foi aplicado o método "escala mínimo máximo" (*Min-Max Scaling*) (HAN; KAMBER; PEI, 2012). O método reescala as variáveis para valores de um a zero, mantendo suas distâncias relativas para evitar que o modelo dê uma importância indevida a variáveis devido à escala das mesmas. O método "escala mínimo máximo" calcula da seguinte maneira:

$$x^*_{ij} = \frac{x_{ij} - min(x_{ij})}{max(x_{ij}) - min(x_{ij})}, \quad j = 1, 2, ...$$

3.1.3 Modelagem - MNIST

Após a aplicação do método PCA e "escala mínimo máximo", a base foi dividida de maneira a preservar a proporção dos *outputs* em base treino e base teste. Essa divisão é dita como divisão estratificada pelo *outputs*, 75% das imagens foram utilizadas para treino e 25% para teste.

O problema em questão possui mais de dois possíveis *outputs*. a lógica utilizada para modelagem foi "um contra todos", portanto, foram criados dez modelos dicotômicos.

A fase de aprendizado do modelo utilizou a base treino e os hiperparâmetros "C" igual a 1 e kernel igual a "rbf". Esses são hiperparâmetros padrões da função utilizada (*sklearn.svm.SVC* da linguagem de programação *python*) e, como o problema é relativamente simples, não houve necessidade de alteração.

3.2 Reconhecimento de Voz

O problema em questão é determinar o gênero do indivíduo com base nas estatísticas de sua fala.

3.2.1 Sanitização - Voz

A base não possui valores faltantes e é balanceada. Todos os *inputs* são estatísticas provenientes da onda sonora emitida por um indivíduo, porém não necessariamente todas as estatísticas são relevantes para o modelo e por isso cada uma delas foi testada.

O primeiro passo para a escolha do teste foi determinar se algum dos *inputs* provém de uma distribuição normal. A normalidade dos dados foi testada pelo teste Jarque-Bera (JARQUE; BERA, 1987), que verifica se a assimetria e a curtose amostrais condizem com de uma distribuição normal, o teste Kolmogorov-Smirnov (SPRENT; SMEETON, 2001), que compara a distância entre a amostra e a distribuição em questão, que, nesse caso, é a distribuição normal, e o teste Shapiro-wilk (SHAPIRO; WILK, 1965), que é o teste clássico para normalidade. Foi rejeitada a hipótese de normalidade em todos os testes para todas as variáveis com um nível de significância de 5%.

3.2.2 Análise exploratória - Voz

Tendo a normalidade rejeitada, para cada variável segmentada por sexo foi aplicado o teste Mann-Whitney U (SPRENT; SMEETON, 2001). Esse teste é um teste não-paramétrico, que tem como hipótese nula que os dados do sexo feminino provêm de uma distribuição com a mesma posição da distribução à qual os dados do sexo masculino provêm. A hipótese nula foi rejeitada para todas as variáveis, com exceção da variável "modindx", portanto a variável "modindx" não agrega informação na distinção de gênero.

O número de dimensões não justifica a aplicação de um método de redução de dimensionalidade. O método "escala mínimo máximo" foi aplicado aos dados, assim como no problema anterior.

3.2.3 Modelagem - Voz

A base final foi dividida em 75% para base treino e 25% para base teste, a divisão manteve as proporções de sexo da amostra.

O aprendizado do modelo se deu com a base treino com hiperparâmetros *default* da função *sklearn.svm.SVC* da linguagem de programação *python* ($C = 1$, *e* $kernel = `rbf'$).

3.3 Classificação de texto

O modelo deve diferenciar e-mails entre "spams" e "não spams"; e-mails classificados como spam são e-mails indesejáveis como propagandas, entre outros. Esse problema de classificação é bem conhecido como um exemplo de quando não se deve utilizar a acurácia como função perda. Como a base é não balanceada, 68% dos e-mails são "não spam". Se o modelo inferir que todos os e-mails são "não spam", ainda assim terá uma acurácia relativamente boa.

3.3.1 Sanitização - Spam

Para a resolução do problema, foram utilizados os *tokens* que, neste caso, são fragmentos de texto em inglês que não contêm espaço e nem números dentro de si para determinar a qual categoria o e-mail pertence (spam e não spam).

A base de dados é uma tabela de frequência dos *tokens* mais comuns dentro dos e-mails. Mais de 90% da base é composta por valores nulos e cerca de 70% dos dados

são referentes a e-mails considerados "não spam". Esta esparcidade é devido ao fato do valor da célula ser referente a frequência do *token* daquela coluna no e-mail, sendo que o conjunto de *tokens* é referente a todos os e-mails amostrados, em outras palavras, é natural que um e-mail não contenha todas as palavras dos demais e-mails. Não há valores faltantes.

Os cinco *tokens* mais comuns dentro da base são "e", "n", "r", "l" e "c", sendo que eles não são palavras reais da língua inglesa. A explicação para esses *tokens* varia de uma escrita errada a um emoticon que perdeu o sentido após a limpeza prévia dos dados, como, por exemplo ":C". Infelizmente, como só foram fornecidos os dados após a limpeza, é impossível discernir a causa desses fragmentos. Como é impossível diferenciar tais resquícios de erros, foram testados dois modelos. Um modelo com a base de treino contendo todos os *tokens* e outro sem qualquer *token* não existente na língua inglesa (caso a maior parte dos fragmentos provenha de algo com significado, esses *tokens* auxiliariam no aprendizado do modelo, porém, se forem meros erros de digitação, o atrapalhariam). Foram detectados 598 *tokens* que não existem na língua inglesa.

Após a remoção de palavras não existentes, outro filtro foi realizado para *stopwords* (GHARATKAR et al., 2017) em ambas as bases. *Stopwords* são palavra que não agregam valor. É necessária sua remoção para evitar que o modelo considere dois e-mails similares simplesmente porque ambos utilizam muitos conectivos como "e", "também", "então" etc. Foram removidas mais de 120 palavras de cada uma das bases.

3.3.2 Análise exploratória - Spam

A base contém a frequência de cada palavra por e-mail e é natural assumir que se um e-mail possui um conjunto de palavras próximo ao de outro e-mail, eles são considerados similares. Porém também deve ser levado em conta no cálculo de similaridade quando dois e-mails contêm palavras em comum muito específicas que quase nunca são utilizadas por outros e-mails.

Para que o modelo possa compreender melhor a distância entre os e-mails, foi calculado o valor *tf-idf* (DALAORAO; SISON; MEDINA, 2019) para cada célula. O valor *tf-idf* é construído para atribuir um peso alto a *tokens* que aparecem com muita frequência e para *tokens* consideradas raros. O cálculo desse valor se dá pela multiplicação da frequência relativa da palavra por e-mail com o log do número de e-mails dividido pelo número de e-mails que contêm a palavra ($log(\frac{N_{e-mails}}{n_{e-mails\ que\ contém}})$).

A base de dados é formada pelas palavras mais frequentes e não todas as palavras,

por isso existe a possibilidade das palavras consideradas raras terem sido removidas da base. Visto essa possibilidade, também foram contruídos modelos que utilizaram em seu aprendizado a frequência relativa (tf) (DALAORAO; SISON; MEDINA, 2019).

O modelo de aprendizagem supervisionada de Máquina de Vetores de Suporte para classificação foi aplicado para quatro diferentes bases, duas com, também, palavras não existentes na língua inglesa (devido à incerteza quanto à natureza desses *tokens* não é possível determinar se são ou não úteis para classificação) e duas com somente palavras existentes na língua inglesa. Entre as duplas de bases, uma utiliza o valor *tfi-df*, que prioriza tanto a frequência da palavra quanto sua raridade e a segunda a frequência relativa.

Apesar de possuir uma grande dimensionalidade, o método PCA não foi aplicado, pois a base possui uma proporção de zeros muito grande, mais de 90%, para casos desse tipo, o método PCA não é recomendado.

3.3.3 Modelagem - Spam

Como esse problema é mais complexo que os anteriores, foi utilizado o "*GridSearch*" do pacote "*sklearn.model_selection*" da línguagem de programação *python*. Esse método consiste em aplicar a validação cruzada dentro de um conjunto de hiperparâmetros informados para determinar qual o melhor conjunto para o problema em questão. O conjunto informado foi "C" variando de 0,1 a 2,0 num passe de 0,1 e os kerneis "polinomial", "linear", "rbf" e "sigmoide"resultando na Tabela 12.

Palavras com erro	Peso	C	Kernel
Removidas	TF	1,9	polinomial
Removidas	TF-IDF	0,7	sigmoide
Mantidas	TF	1,9	polinomial
Mantidas	TF-IDF	0,7	sigmoide

Tabela 13: Bases treinos - *Text*.
Fonte: autoria própria.

Todas as bases tratadas foram divididas estratificadamente pela classe do e-mail em base treino e teste, sendo que a base treino contém 75% dos e-mails e teste, 25%. Após a divisão, o aprendizado dos modelos foi realizado com os hiperparâmetros e as bases treino provenientes da base tratada correspondente e posteriormente testados com a base teste também correpondente.

4 Resultados

4.1 Classificação de imagem

A base treino possui 31.500 imagens, sendo que cada imagem possuía 87 *inputs*. O modelo levou cerca de 41,63 segundos para finalizar seu treino.

Após o treino, o modelo recebeu os *inputs* da base teste, que contém 10.500 imagens de dígitos escritos à mão, para tentar reconhecer o mesmo. O *output* do modelo, juntamente com o resultado real, foi usado para o cálculo do *f1-score* registrando os seguintes valores:

Digito	*f1-score*	Número de imagens
0	99%	1033
1	99%	1171
2	97%	1044
3	97%	1088
4	97%	1018
5	98%	949
6	98%	1034
7	97%	1100
8	97%	1016
9	97%	1047

Tabela 14: *f1-score - Digits.*
Fonte: autoria própria.

Com o *f1-score* para cada *output*, é possível calcular o *f1-weighted* (SASAKI, 2007) pela seguinte fórmula:

$$f1\text{-}weighted = \frac{\sum_{i=0}^{l} f1\text{-}score_i n_i}{\sum_{i=1}^{l} n_i}$$

$$l = N\ outputs; \quad n_i = N\ obs.\ no\ output\ i;$$

O modelo, após o treino, obteve um *f1-weighted* igual a 98%. Além do *f1-weighted* foi também calculado a acurácia e seu resultado apresentado no gráfico seguinte.

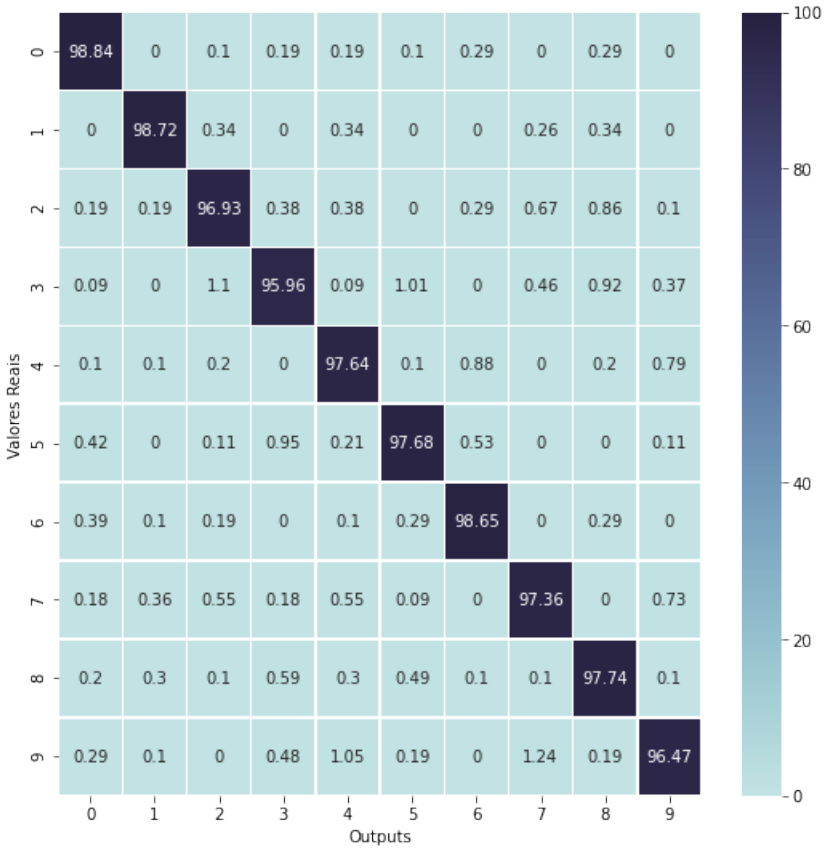


Figura 31: Acurácia do modelo de classificação de imagem.
Fonte: autoria própria.

As células da Figura 31 contém a frequência relativa dos resultados do modelo (eixo X) para os reais valores da variável respostas (eixo Y). As células na diagonal principal (diagonal da esquerda em cima para direita em baixo) contém a porcentagem de classificações corretas. O modelo obteve excelentes resultados, um *f1-weighted* de 98% e uma acurácia de pelo menos 95% para cada classificação.

4.2 Reconhecimento de voz

A base treino utilizada para o aprendizado do modelo possui 2.374 áudios, sendo 1187 do sexo masculino e 1188 do sexo feminino; o treino do modelo levou cerca de 0,05 segundos e, quando aplicado na base teste de 792 áudios, obteve um *f1-score* de 98%.

4.3 Classificação de texto

Esse problema obteve quatro modelos criados a partir de diferentes bases e diferentes hiperparâmetros, os resultados foram sumarizados e relatados abaixo:

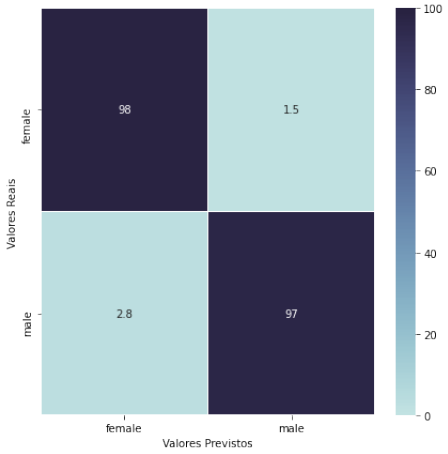


Figura 32: Acurácia do modelo de Reconhecimento de voz.
Fonte: autoria própria.

Id	Palavras com erro	Peso	C	gamma	Kernel	*f1-weighted*	Tempo treino
1	Removidas	TF	1,9	scale	polinomial	91%	14,86 s
2	Removidas	TF-IDF	0,7	scale	sigmoide	96%	36,49 s
3	Mantidas	TF	1,9	scale	polinomial	93%	16,14 s
4	Mantidas	TF-IDF	0,7	scale	sigmoide	96%	21,66 s

Tabela 15: Resumo dos modelos - *Text*.
Fonte: autoria própria.

De acordo com o *f1-weighted*, o modelo escolhido seria o de identificação número 4 ou 2, porém, como o objetivo do modelo é identificar quais e-mails são de fato spam, a escolha deve ser associada também com acurácia da classe spam. Nos gráficos abaixo, os e-mails spam são representados pelo número 1 e não spam, pelo número 0.

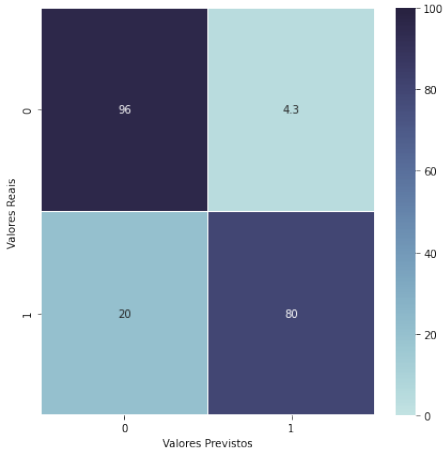


Figura 33: Acurácia do modelo Nº 1.
Fonte: autoria própria.

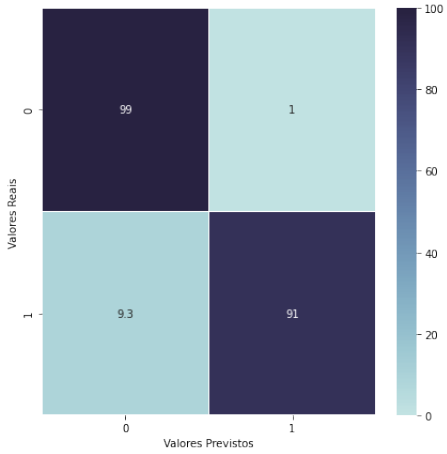


Figura 34: Acurácia do modelo Nº 2.
Fonte: autoria própria.

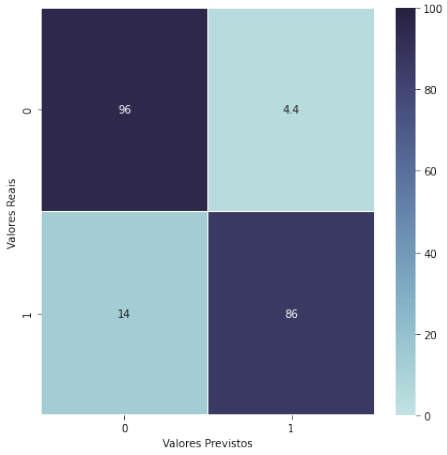


Figura 35: Acurácia do modelo Nº 3.
Fonte: autoria própria.

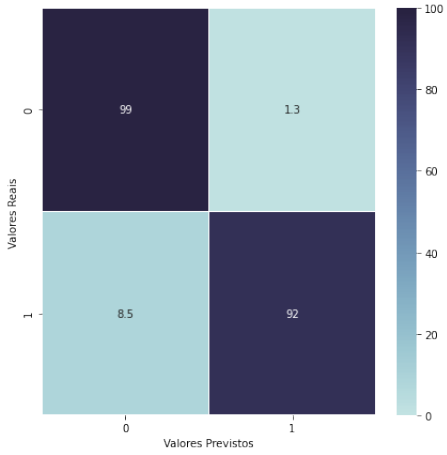


Figura 36: Acurácia do modelo Nº 4.
Fonte: autoria própria.

O modelo que teve a melhor performance entre todos, de acordo com o *f1-weighted* e a acurácia de spam, foi o modelo número 4. Todos os modelos tiveram seu aprendizado realizado com base em 3.473 e-mails e testados com 1.158 e-mails, sendo que as bases com *tokens* removidos possuíam 2.275 *inputs* e o restante das bases 2.866. O modelo número 4 ter sido o que teve o melhor resultado indica que os *tokens* que não constam na língua inglesa de fato foram importantes para a classificação do e-mail e que, apesar de não possuir todas as palavras, a atribuição de peso aos *inputs* com base na sua raridade auxiliou o modelo em seu aprendizado.

Conclusão

Conforme proposto, neste trabalho foi aplicada a Máquina de Vetores de Suporte para classificação em três diferentes problemas, cada um possuindo uma ou mais dificuldades específicas quanto ao processamento dos dados e/ou treinamento do modelo.

No primeiro problema, a dificuldade encontrada foi que, devido ao truque de kernel, o custo do modelo ao trabalhar com diversos *inputs* é muito alto. Para esses casos, quando possível, a melhor solução é aplicar um método de redução de dimensionalidade, como o PCA. Apesar da dificuldade, o modelo atingiu um *f1-weighted* de 98%.

No segundo, o aprendizado do modelo foi realizado com base em estatísticas de um banco de áudio. Nesse caso, a grande dificuldade foi lidar com a possível redundância de informações, já que é natural assumir que alguns dados possuam correlações muito altas (como no caso dos *inputs* "Q25", "median"e "Q75", que foram mantidos para avaliar o comportamento do modelo quanto a essa dificuldade). Ainda assim, o modelo foi capaz de obter uma acurácia média de gênero de mais de 97%.

O último problema possui diversos fatores adversos: o banco possui 90% dos dados iguais a zero; a base tratada possuía três mil colunas e, pela quantidade zeros, não foi possível aplicar o PCA; os dados eram desbalanceados numa proporção de, aproximadamente, 68/32. Apesar disso, o modelo foi capaz de atingir um *f1-weighted* de 96% e uma acurácia para classificação de "spam"de 92%.

Cada problema exigiu uma abordagem diferente, o que possibilitou demonstrar a sua capacidade de adaptação. A partir da metodologia aplicada, abordamos todas as fases da modelagem, desde o tratamento dos dados à interpretação dos resultados. Dessa maneira, acreditamos ter atingido nosso propósito de apresentar uma introdução ao método, possibilitando que o leitor seja capaz de resolver problemas de classificação e de se aprofundar na literatura sobre o modelo quando necessário ou desejado.

Referências

ANIMATRONBOT. *MNIST Digit Recognizer*. 2018. Disponível em: ⟨https://www.kaggle.com/datasets/animatronbot/mnist-digit-recognizer⟩.

BECKER, K. *Gender Recognition by Voice*. 2016. Disponível em: ⟨https://www.kaggle.com/datasets/primaryobjects/voicegende⟩.

BISWAS, B. *Email Spam Classification Dataset CSV*. 2020. Disponível em: ⟨https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv⟩.

BOLDRINI, J. l. et al. *Álgebra Linear*. [S.l.]: editora HARBRA ltda., 1986. v. 3º edição. 99-100 p. ISBN 9788529402022.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. 1992.

BOWNE-ANDERSON, H. *Supervised Learning with scikit-learn*. 2017. Disponível em: ⟨https://learn.datacamp.com/courses/supervised-learning-with-scikit-learn⟩.

CALLIOLI, C. A.; DOMINGUES, H. H.; COSTA, R. C. F. *Álgebra linear e aplicações*. [S.l.]: ATUAL EDITORA, 1990. v. 6º edição. 158 - 163 p. ISBN 8570562977.

CALLIOLI, C. A.; DOMINGUES, H. H.; COSTA, R. C. F. *Álgebra linear e aplicações*. [S.l.]: ATUAL EDITORA, 1990. v. 6º edição. 298 - 304 p. ISBN 8570562977.

DALAORAO, G. A.; SISON, A. M.; MEDINA, R. P. Integrating collocation as tf-idf enhancement to improve classification accuracy. In: *2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*. [S.l.: s.n.], 2019.

FARIAS, D. M.; KONZEN, P. H. d. A.; SOUZA, R. R. Álgebra linear um livro colaborativo. p. 11 – 12, 2020.

FARIAS, D. M.; KONZEN, P. H. d. A.; SOUZA, R. R. Álgebra linear um livro colaborativo. p. 72, 2020.

GHARATKAR, S. et al. Review preprocessing using data cleaning and stemming technique. In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. [S.l.: s.n.], 2017.

HAN, J.; KAMBER, M.; PEI, J. *Data Mining: Concepts and Techniques*. [S.l.]: Elsevier, 2012. Third Edition. ISBN 9780123814807.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2008. Second Edition. 129-130 p.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2008. Second Edition. 132 - 133 p.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2008. Second Edition. 423 - 425 p.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2008. Second Edition. 431 p.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The Elements of Statistical Learning*. [S.l.]: Springer, 2008. Second Edition. 419 - 421 p.

JARQUE, C. M.; BERA, A. K. *A Test for Normality of Observations and Regression Residuals)*. [S.l.: s.n.], 1987.

JOLLIFFE, I. T. *Principal Component Analysis*. [S.l.]: Springer, 2017. Second Edition.

LECUN, Y.; CORTES, C.; BURGES, C. J. *THE MNIST DATABASE*. Disponível em: ⟨http://yann.lecun.com/exdb/mnist/⟩.

SASAKI, Y. The truth of the f-measure. 2007.

SHAPIRO, Y. S.; WILK, M. B. An analysis of variance test for normality (complete samp1es). 1965.

SPRENT, P.; SMEETON, N. *Applied nonparametric statistical methods*. [S.l.: s.n.], 2001. Third Edition. ISBN 1584881453.

WINSTON, P. H. *16. Learning: Support Vector Machines*. 2010. Disponível em: ⟨https://www.youtube.com/watch?v=_PwhiWxHK8o⟩.

Anexo

Nesta seção, está exposto todos os códigos usados da línguagem de programação *python* para gerar as imagens, exemplos e análises presentes no trabalho. Porém antes é necessário executar o código abaixo para importação dos pacotes utilizados e importação das bases.

Arquivo .py que indica o caminho das bases:

```python
from pathlib import Path

# Acrescente abaixo o Path das bases de dados
paths = ['C:/Users/pedro/Documents/Unb/2021-2/TCC 2/TCC Jupyter/data',
         'C:/Users/PedroMoura/Documents/TCC/TCC 2/TCC Jupyter/data']

# Procurando Path existente
DATA_PATH = False
for path in paths:
    if Path(path).exists():
        DATA_PATH = Path(path)
        break

# Caso não haja Path existente se assume o atual
if not DATA_PATH:
    DATA_PATH = Path.cwd()

# Outros Paths
DATA_PATH_RAW = DATA_PATH / 'raw'
DATA_PATH_WRANGLE = DATA_PATH / 'wrangle'
DATA_PATH_RESULTS = DATA_PATH / 'results'
```

Fonte: autoria própria.

Importação dos módulos utilizados:

```python
# Acesso aos meu módulos
import sys
from pathlib import Path
sys.path.insert(1, Path.cwd().parents[0].as_posix())

# Pacotes padrões
import pandas as pd
import numpy as np
from tqdm import tqdm
from datetime import datetime
from random import sample, seed

# Pacotes de visualização
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image

# Testes
from scipy.stats import shapiro, jarque_bera, kstest, mannwhitneyu

# Pré-processamento
import enchant
from nltk.corpus import stopwords
from sklearn import preprocessing
from sklearn.decomposition import PCA

# Modelo
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix

# Meus módulos
from config import *
```

Fonte: autoria própria.

O *output* dos códigos, quando existente, está disposto abaixo do mesmo para faci-

litar o entendimento.

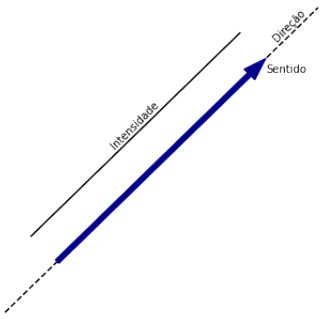
.1 Referencial teórico

.1.1 Imagens dos conceitos algébricos

Figura 1:

```
plt.figure(figsize=(7,7))

# Figuras
ax = sns.lineplot(x=[0.5, 3.5], y=[0.5, 3.5], linestyle='--', color='black', zorder=1)
sns.lineplot(x=[0.75, 2.75], y=[1.25, 3.25], color='black', zorder=1)
plt.arrow(1, 1, 2, 2, color='darkblue', width=0.05, length_includes_head=True, zorder=2)

# Escritas
plt.annotate('Direção', xy=(3.05, 3.15), rotation=45, ha='left', va='bottom')
plt.annotate('Intensidade', xy=(1.5, 2.1), rotation=45, ha='left', va='bottom')
plt.annotate('Sentido', xy=(3, 2.9), ha='left', va='center')

# Configurações
ax.set_axis_off()
plt.xlim(0,4)
plt.ylim(0,4)

plt.show()
```

Fonte: autoria própria.

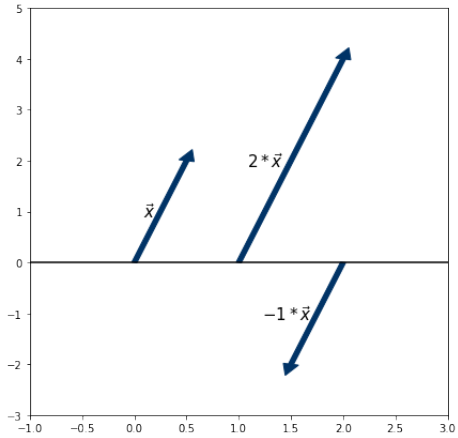


Fonte: autoria própria.

Figura 2:

```
plt.figure(figsize=(7,7))

# Figuras
sns.lineplot(x=[-5, 5], y=[0,0], color='black')
plt.arrow(0,0,0.5,2, color='#003366', width=0.05)
plt.arrow(1,0,1,4, color='#003366', width=0.05)
plt.arrow(2,0,-0.5,-2, color='#003366', width=0.05)

# Escritas
plt.annotate(r'$\vec{x}$', xy=(0.2,1), ha='right', va='center', size=15)
plt.annotate(r'$-1*\vec{x}$', xy=(1.7,-1), ha='right', va='center', size=15)
plt.annotate(r'$2*\vec{x}$', xy=(1.25,2), ha='center', va='center', size=15)

# Configurações
plt.xlim(-1, 3)
plt.ylim(-3, 5)

plt.show()
```

Fonte: autoria própria.

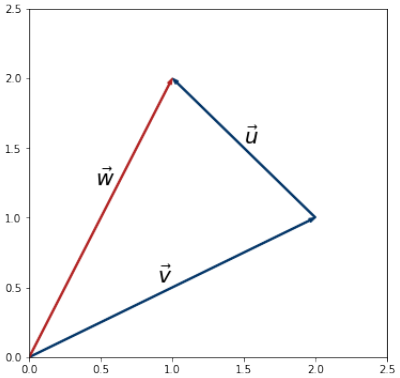


Fonte: autoria própria.

Figura 3:

```
plt.figure(figsize=(6, 6))

# Figuras
plt.arrow(0, 0, 2, 1, color='#003366', width=0.01, length_includes_head=True)
plt.arrow(2, 1, -1, 1, color='#003366', width=0.01, length_includes_head=True)
plt.arrow(0, 0, 1, 2, color='firebrick', width=0.01, length_includes_head=True)

# Escritas
plt.annotate(r'$\vec{u}$', xy=(1.5, 1.5), ha='left', va='bottom', size=20)
plt.annotate(r'$\vec{v}$', xy=(1, .5), ha='right', va='bottom', size=20)
plt.annotate(r'$\vec{w}$', xy=(0.6, 1.2), ha='right', va='bottom', size=20)

# Configurações
plt.xlim(0, 2.5)
plt.ylim(0, 2.5)

plt.show()
```

Fonte: autoria própria.

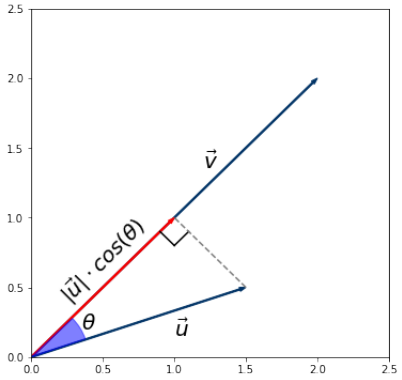


Fonte: autoria própria.

Figura 4:

```
fig, ax = plt.subplots(1, 1, figsize=(6, 6))

# Figuras
ax.arrow(0, 0, 2, 2, color='#003366', width=0.01, length_includes_head=True)
ax.arrow(0, 0, 1, 1, color='red', width=0.01, length_includes_head=True)
ax.arrow(0, 0, 1.5, 0.5, color='#003366', width=0.01, length_includes_head=True)

sector = Wedge((0, 0), .4, 19, 44, color="blue", alpha=0.5)
ax.add_artist(sector)

ax.plot([0.9, 1], [0.9, 0.8], color='black')
ax.plot([1, 1.1], [0.8, 0.9], color='black')
ax.plot([1.5, 1], [0.5, 1], '--', color='gray')

ax.annotate(r'$\vec{u}$', xy=(1, 0.15), size=20)
ax.annotate(r'$\theta$', xy=(0.35, 0.2), size=20)
ax.annotate(r'$\vec{v}$', xy=(1.2, 1.35), size=20)
ax.annotate(r'$|\vec{u}| \cdot cos(\theta)}$', xy=(0.17, 0.4), size=20, rotation=45)

# Configurações
plt.xlim(0, 2.5)
plt.ylim(0, 2.5)

plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

.1.2 Máquina de vetores de suporte para classificação

Dados contidos na tabela 1:

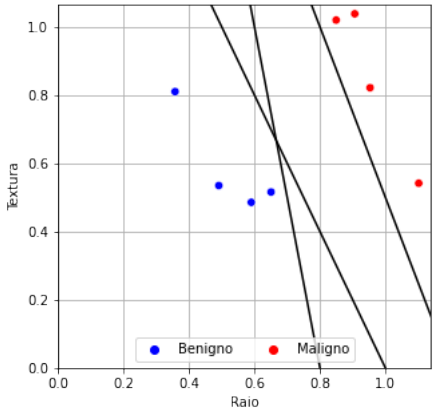
```
# Função para criação de dados
def dados(r, media, desvio, n=4):
    np.random.seed(r)
    return np.random.normal(media, desvio, n)

# Dataframe
palette_tumor = {'Maligno': 'red', 'Benigno': 'blue'}
benignos = pd.DataFrame({'Raio': dados(13, 0.5, 0.2), 'Textura': dados(14, 0.5, 0.2), 'Tipo': 'Benigno'})
malignos = pd.DataFrame({'Raio': dados(15, 1, 0.3), 'Textura': dados(16, 1, 0.3), 'Tipo': 'Maligno'})
df = benignos.append(malignos).reset_index(drop=True)

# Divisão de inputs e outpus
X = df.drop(columns='Tipo')
y = df['Tipo']
```

Fonte: autoria própria.

Figura 5:

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)

sns.lineplot(x=[0, 1], y=[2, 0], color='black', zorder=1)
sns.lineplot(x=[0, 1.2], y=[3, 0], color='black', zorder=1)
sns.lineplot(x=[0, 0.8], y=[4, 0], color='black', zorder=1)

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

Modelagem para criar a Figura 6:

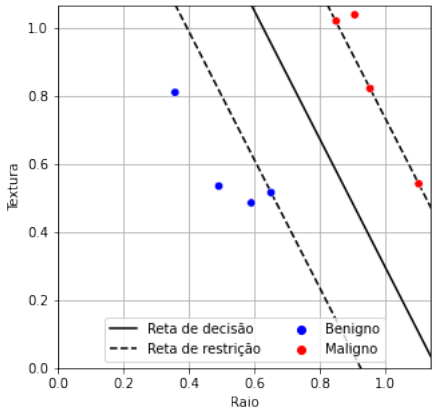
```
# Criação do modelo
model = svm.SVC(kernel='linear', C=float('inf'))
model.fit(X, y)


# Criação das retas (restrições e decisão)
w = model.coef_[0]
b = model.intercept_[0]
x0 = np.linspace(0, 3, 200)
decision_boundary = -w[0]/w[1] * x0 - b/w[1]

margin = 1/w[1]
gutter_up = decision_boundary + margin
gutter_down = decision_boundary - margin
```

Fonte: autoria própria.

Figura 6:

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)

sns.lineplot(x=x0, y=decision_boundary, color='black', zorder=1, label='Reta de decisão')
sns.lineplot(x=x0, y=gutter_up, color='black', linestyle='--', zorder=1, label='Reta de restrição')
sns.lineplot(x=x0, y=gutter_down, color='black', linestyle='--', zorder=1)

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.



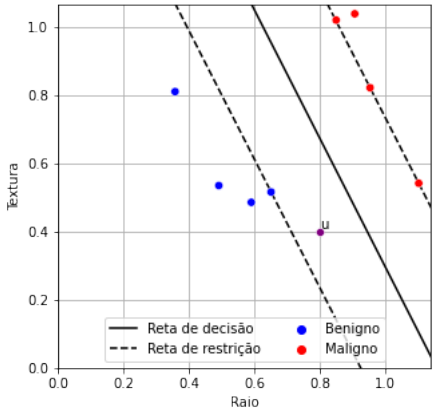
Fonte: autoria própria.

Figura 7:

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)

sns.lineplot(x=x0, y=decision_boundary, color='black', zorder=1, label='Reta de decisão')
sns.lineplot(x=x0, y=gutter_up, color='black', linestyle='--', zorder=1, label='Reta de restrição')
sns.lineplot(x=x0, y=gutter_down, color='black', linestyle='--', zorder=1)

# Novo ponto u
ponto_u = pd.DataFrame({'x': [0.8], 'y': [0.4]})
sns.scatterplot(x='x', y='y', data=ponto_u, color='purple')
plt.annotate('u', xy=(ponto_u.loc[0].tolist()), ha='left', va='bottom')

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.

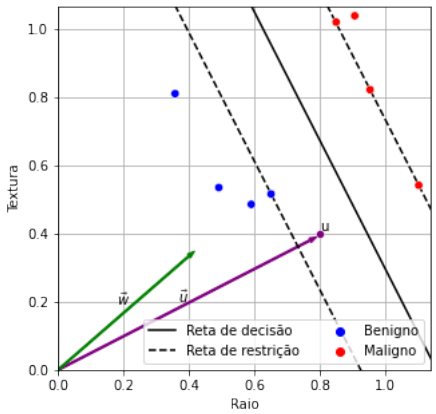
Figura 8:



Fonte: autoria própria.

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)

sns.lineplot(x=x0, y=decision_boundary, color='black', zorder=1, label='Reta de decisão')
sns.lineplot(x=x0, y=gutter_up, color='black', linestyle='--', zorder=1, label='Reta de restrição')
sns.lineplot(x=x0, y=gutter_down, color='black', linestyle='--', zorder=1)

# Novo ponto u
sns.scatterplot(x='x', y='y', data=ponto_u, color='purple')
plt.annotate('u', xy=(ponto_u.loc[0].tolist()), ha='left', va='bottom')

# Vetores
plt.arrow(0, 0, 0.77, 0.38, width = 0.005, color='purple')
plt.arrow(0, 0, 0.4, 0.334, width = 0.005, color='green')
plt.annotate(r'$\vec{w}$', xy=(0.22,0.18), ha='right', va='bottom')
plt.annotate(r'$\vec{u}$', xy=(0.4,0.19), ha='right', va='bottom')

# Configurações
plt.legend(loc="lower right", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.

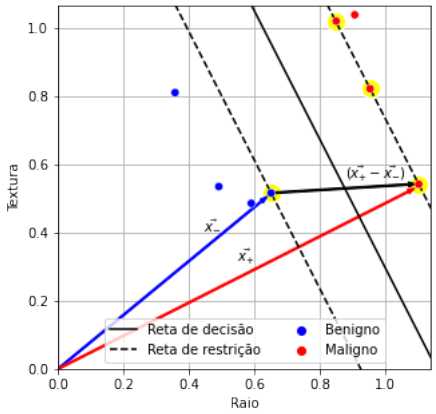


Fonte: autoria própria.

Figura 9:

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)

sns.lineplot(x=x0, y=decision_boundary, color='black', zorder=1, label='Reta de decisão')
sns.lineplot(x=x0, y=gutter_up, color='black', linestyle='--', zorder=1, label='Reta de restrição')
sns.lineplot(x=x0, y=gutter_down, color='black', linestyle='--', zorder=1)

# Pontos Destacados
sns.scatterplot(x='Raio', y='Textura', s=200, color='yellow', data=df.loc[((df['Raio']>0.6) & (df['Tipo'] == 'Benigno')) |
                                                                          ((df['Textura']<1.02) & (df['Tipo'] == 'Maligno'))],
                zorder=1, legend=False)

# Vetores
plt.arrow(0, 0, 0.622, 0.492, width = 0.005, color='blue')
plt.annotate(r'$\vec{x_{-}}$', xy=(0.5, 0.4), ha='right', va='bottom')

plt.arrow(0, 0, 1.07, 0.52, width = 0.005, color='red')
plt.annotate(r'$\vec{x_{+}}$', xy=(0.6, 0.3), ha='right', va='bottom')

plt.arrow(0.650753, 0.515837, 0.42, 0.025, width = 0.005, color='black')
plt.annotate(r'$(\vec{x_{+}} - \vec{x_{-}})$', xy=(0.97, 0.54), ha='center', va='bottom')

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.



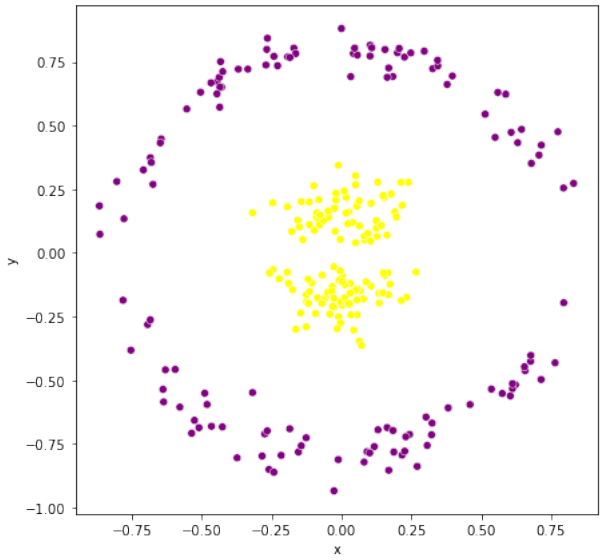
Fonte: autoria própria.

Código de criação dos dados e da Figura 10:

Figura 11

```
# Função para criar dados
def circulo(R):
    a = np.random.uniform()
    b = np.sqrt(abs(a**2 - 1))

    a=sample([a, -a], 1)[0]
    b=sample([b, -b], 1)[0]
    return a*R, b*R

# Colunas
x = []
y = []
classe = []

# Criação dos dados
for i in range(0, 130):
    np.random.seed(i)
    a, b = circulo(np.random.normal(0.8, 0.05))
    x.append(a)
    y.append(b)
    classe.append(1)

    a, b = circulo(np.random.normal(0.2, 0.07))
    x.append(a)
    y.append(b)
    classe.append(2)

# Dataframe
kernel = pd.DataFrame({'x':x, 'y':y, 'classe':classe})

# Plot
plt.figure(figsize = (7, 7))

sns.scatterplot(x='x', y='y', hue='classe', palette={1:'purple', 2:'yellow'}, data=kernel, legend=False)

plt.show()
```

Fonte: autoria própria.

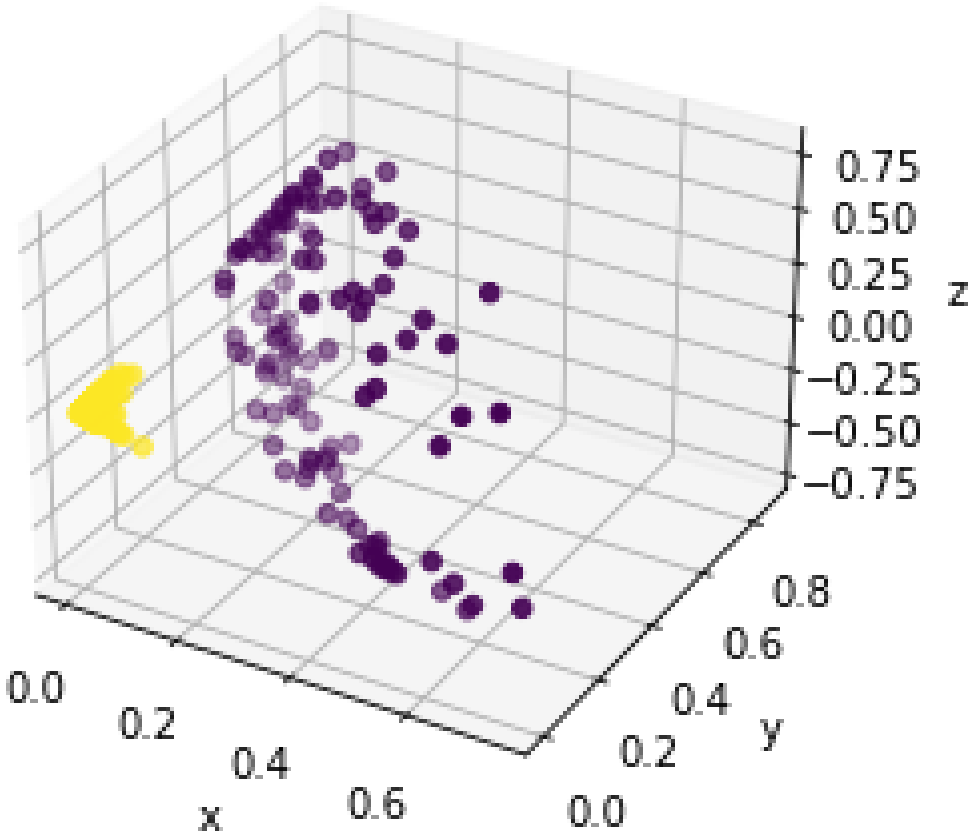


Fonte: autoria própria.

```
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

# Transformação dos dados
x = kernel['x']**2
y = kernel['y']**2
z = 2 * kernel['x'] * kernel['y']
c = kernel['classe']

# Figures
ax.scatter(x, y, z, c=c)

# Configurações
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

plt.show()
```

Fonte: autoria própria.

Código de criação dos dados e da Figura 12:

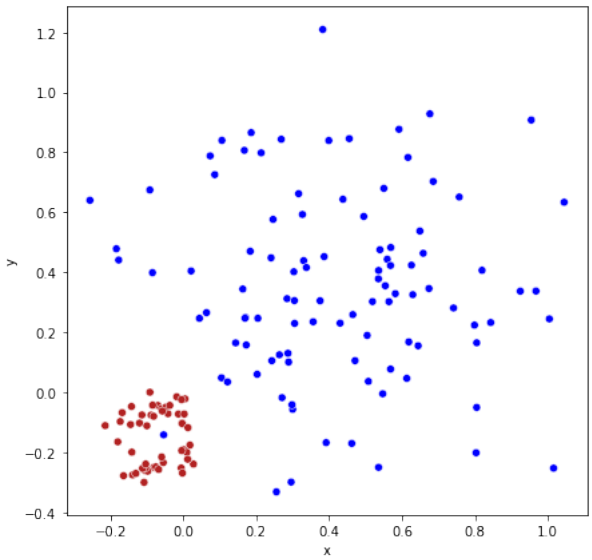


Fonte: autoria própria.

Figura 13:

```
# Função para criar dados
def circulo(R, i):
    np.random.seed(i)
    a = np.random.uniform()
    b = np.sqrt(abs(a**2 - 1))

    seed(i)
    a=sample([a, -a], 1)[0]
    b=sample([b, -b], 1)[0]
    return a*R, b*R

# Dados
n=100
palette_tumor = {'Maligno': 'red', 'Benigno': 'blue'}
c_1 = pd.DataFrame({'x': dados(13, 0.4, 0.3, n=n), 'y': dados(14, 0.4, 0.3, n=n), 'classe': 1})

# Colunas
x = []
y = []
classe = []

# Criação dos dados
for i in range(0, 50):
    np.random.seed(i)
    c = np.random.normal(0.1, 0.03)
    a, b = circulo(c, i)
    x.append(a - 0.08)
    y.append(b - 0.15)

c_2 = pd.DataFrame({'x': x, 'y': y, 'classe': 2})
df_over = c_1.append(c_2)

# Divisão de inputs e outputs
Xo = df_over.drop(columns='classe')
yo = df_over['classe']

# Gráficos
plt.figure(figsize = (7, 7))

sns.scatterplot(x='x', y='y', hue='classe', palette={1:'blue', 2:'firebrick'}, data=df_over, legend=False)

plt.show()
```

Fonte: autoria própria.

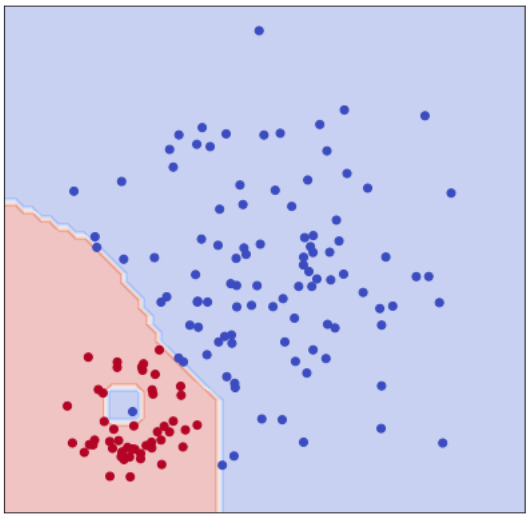


Fonte: autoria própria.

Figura 14:

```
# Separando os dados em inputs, outputs e mesh
X = df_over[['x', 'y']].to_numpy()
y = df_over['classe'].to_numpy()
h = .02

# Modelo
modelo = svm.SVC(kernel='rbf', C=float('inf')).fit(X, y)

# Criação do mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Gráfico
plt.figure(figsize=(7, 7))

# Figuras
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)

# Configurações
plt.xlim(-0.2, 1)
plt.ylim(-0.2, 1)
plt.xticks(())
plt.yticks(())

plt.show()
```

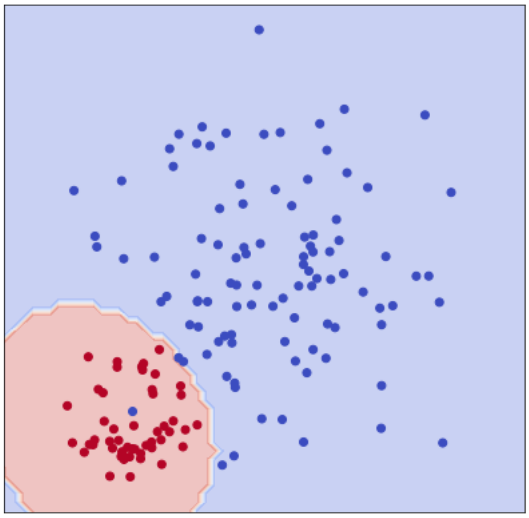
Fonte: autoria própria.



Fonte: autoria própria.

```
# Separando os dados em inputs, outputs e mesh
X = df_over[['x', 'y']].to_numpy()
y = df_over['classe'].to_numpy()
h = .02

# Modelos
modelo = svm.SVC(kernel='rbf', C=1).fit(X, y)

# Criação do mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Gráfico
plt.figure(figsize=(7, 7))

# Figuras
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)

# Configurações
plt.xlim(-0.2, 1)
plt.ylim(-0.2, 1)
plt.xticks(())
plt.yticks(())

plt.show()
```

Fonte: autoria própria.

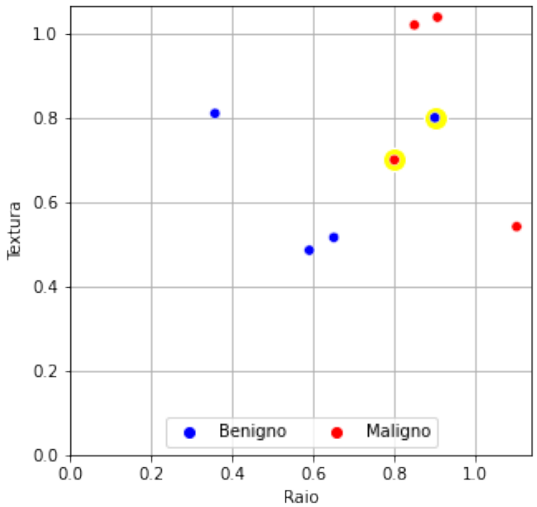


Fonte: autoria própria.

Figura 15:

```
# Dados
df_lap = df.copy()
df_lap.loc[6, ['Textura', 'Raio']] = [0.8, 0.8]
df_lap.loc[2, ['Textura', 'Raio']] = [0.8, 0.9]

# Gráficos
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df_lap, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)
sns.scatterplot(x=[0.9, 0.8], y=[0.8, 0.8], s=200, color='yellow', zorder=1, legend=False)

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])

plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

Código utilizado para criar o modelo da Figura 16:

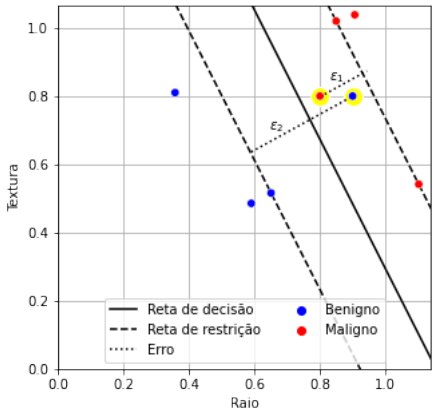
```
# Criação do modelo
X = df_lap.drop(columns='Tipo')
y = df_lap['Tipo']
model = svm.SVC(kernel='linear', C=20)
model.fit(X, y)

# Criação das retas (restrições e decisão)
w = model.coef_[0]
b = model.intercept_[0]
x0 = np.linspace(0, 3, 200)
decision_boundary = -w[0]/w[1] * x0 - b/w[1]

margin = 1/w[1]
gutter_up = decision_boundary + margin
gutter_down = decision_boundary - margin

xe1 = np.linspace(0.59, 0.9, 200)
xe2 = np.linspace(0.8, 0.9435, 200)

# Retas erro
erro1 = w[1]/w[0] * xe1 + 0.325
erro2 = w[1]/w[0] * xe2 + 0.376
```

Fonte: autoria própria.

Figura 16:

```
plt.figure(figsize=(5, 5))

# Figuras
ax=sns.scatterplot(x='Raio', y='Textura', hue='Tipo', s=40, palette=palette_tumor, data=df_lap, zorder=2)
xlim = ax.get_xlim()
ylim = ax.get_ylim()
plt.grid()
ax.set_axisbelow(True)
sns.scatterplot(x=[0.8, 0.9], y=[0.8, 0.8], s=200, color='yellow', zorder=1, legend=False)

# Retas decisão e restrição
sns.lineplot(x=x0, y=decision_boundary, color='black', zorder=1, label='Reta de decisão')
sns.lineplot(x=x0, y=gutter_up, color='black', linestyle='--', zorder=1, label='Reta de restrição')
sns.lineplot(x=x0, y=gutter_down, color='black', linestyle='--', zorder=1)

# Retas erro
sns.lineplot(x=xe1, y=erro1, color='black', zorder=1, linestyle='dotted', label='Erro')
sns.lineplot(x=xe2, y=erro2, color='black', zorder=1, linestyle='dotted', legend=False)

# Escritas
plt.annotate(r'$\varepsilon_1$', xy=(0.85,0.83), ha='center', va='bottom')
plt.annotate(r'$\varepsilon_2$', xy=(0.665,0.685), ha='center', va='bottom')

# Configurações
plt.legend(loc="lower center", ncol=2)
plt.xlim(0, xlim[1])
plt.ylim(0, ylim[1])
plt.show()
```

Fonte: autoria própria.

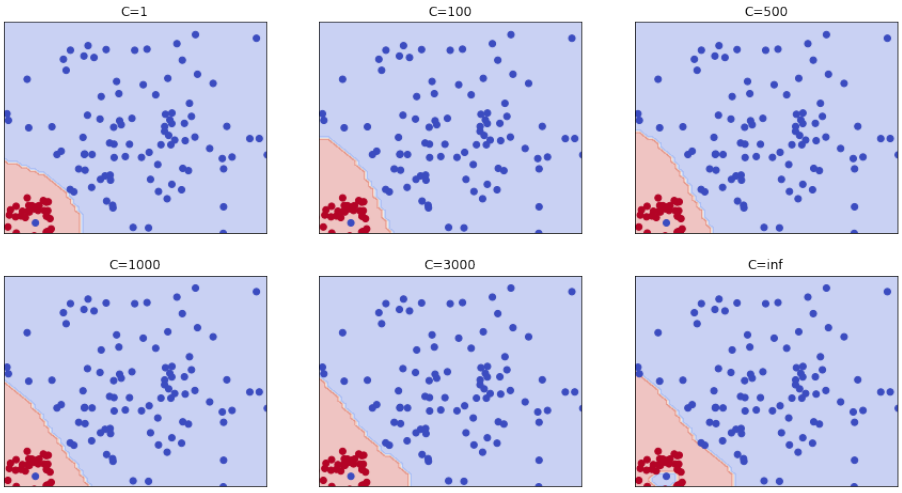


Fonte: autoria própria.

Figura 17:

```
fig, ax = plt.subplots(2, 3, figsize=(15, 8))

# Separação dos dados em inputs, outputs e mesh
X = df_over[['x', 'y']].to_numpy()
y = df_over['classe'].to_numpy()
h = .02

# Vários gráficos
for i, c in enumerate([1, 100, 500, 1000, 3000, float('inf')]):

    # Modelo
    modelo = svm.SVC(kernel='rbf', C=c).fit(X, y)

    # Criação do mesh
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = modelo.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Figuras
    ax[i//3, i%3].contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)
    ax[i//3, i%3].scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)

    # Configurações
    ax[i//3, i%3].set_xlim(-0.2, 1)
    ax[i//3, i%3].set_ylim(-0.2, 1)
    ax[i//3, i%3].set_xticks(())
    ax[i//3, i%3].set_yticks(())
    ax[i//3, i%3].set_title('C={}'.format(c))


plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

.2 Conjunto de dados

.2.1 Classificação de imagens

Importação da base de dados de dígitos:

```
# Máximo de coluns
pd.set_option('display.max_columns', 10)

# Importação da base de digitos
digits = pd.read_csv(DATA_PATH_RAW / 'digits.csv')

# Descrição base
print('Dimensão:', digits.shape)
print('N duplicatas', digits.duplicated().sum())
print('Mín:', digits.min().min(), 'Máx:', digits.max().max())
print('N de zeros:', (digits == 0).sum().sum(),
      'Porcentagem:', round(100 * (digits == 0).sum().sum() / (digits.shape[0] * digits.shape[1]), 2))
print('N de missings:', (digits.isna()).sum().sum(),
      'Porcentagem:', round(100 * (digits.isna()).sum().sum() / (digits.shape[0] * digits.shape[1]), 2))

digits.head()
```

Fonte: autoria própria.

```
Dimensão: (42000, 785)
N duplicatas 0
Mín: 0 Máx: 255
N de zeros: 26625444 Porcentagem: 80.76
N de missings: 0 Porcentagem: 0.0
```

	label	pixel0	pixel1	pixel2	pixel3	...	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	...	0	0	0	0	0
1	0	0	0	0	0	...	0	0	0	0	0
2	1	0	0	0	0	...	0	0	0	0	0
3	4	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	...	0	0	0	0	0

Fonte: autoria própria.

Código de divisão entre *inputs* e *outputs*:

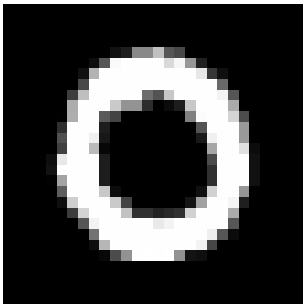
```
# Separação de dados
y = digits['label'].copy()
X = digits.drop(columns = ['label']).copy()
```

Fonte: autoria própria.

Código para visualização das imagens com a resolução de 28x28 pixels:

```
# Index da imagem a ser recriada
i=1

# Print do label da imagem
print(y[i], '\n\n')

# Lista contendo o valor de todos os pixels e matrix que gerará a imagem
lista_image = X.loc[i].tolist()
matrix_image = []

# Preenchendo a Matrix mantendo a relação 28x28
while len(lista_image) >= 28:
    matrix_image.append(lista_image[:28])
    lista_image = lista_image[28:]
matrix_image = np.array(matrix_image)

# Print da imagem
img = Image.fromarray(matrix_image)
img.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

Visualização da imagem no formato de matriz:

```
# Configuração para visualizar os dados
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)

# Matrix
pd.DataFrame(matrix_image)
```

Fonte: autoria própria.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	18	30	137	137	192	86	72	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	13	86	250	254	254	254	254	217	246	151	32	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	16	179	254	254	254	254	254	254	254	254	254	231	54	15	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	72	254	254	254	254	254	254	254	254	254	254	254	254	104	0	0	0	0	0	0	0
8	0	0	0	0	0	0	61	191	254	254	254	254	254	109	83	199	254	254	254	254	243	85	0	0	0	0	0	0
9	0	0	0	0	0	0	172	254	254	254	202	147	147	45	0	11	29	200	254	254	254	171	0	0	0	0	0	0
10	0	0	0	0	0	1	174	254	254	89	67	0	0	0	0	0	0	128	252	254	254	212	76	0	0	0	0	0
11	0	0	0	0	0	47	254	254	254	29	0	0	0	0	0	0	0	83	254	254	254	153	0	0	0	0	0	0
12	0	0	0	0	0	80	254	254	240	24	0	0	0	0	0	0	0	25	240	254	254	153	0	0	0	0	0	0
13	0	0	0	0	0	64	254	254	186	7	0	0	0	0	0	0	0	0	166	254	254	224	12	0	0	0	0	0
14	0	0	0	0	14	232	254	254	254	29	0	0	0	0	0	0	0	0	75	254	254	254	17	0	0	0	0	0
15	0	0	0	0	18	254	254	254	254	29	0	0	0	0	0	0	0	0	48	254	254	254	17	0	0	0	0	0
16	0	0	0	0	2	163	254	254	254	29	0	0	0	0	0	0	0	0	48	254	254	254	17	0	0	0	0	0
17	0	0	0	0	0	94	254	254	254	200	12	0	0	0	0	0	0	0	16	209	254	254	150	1	0	0	0	0
18	0	0	0	0	0	15	206	254	254	202	66	0	0	0	0	0	0	21	161	254	254	245	31	0	0	0	0	0
19	0	0	0	0	0	0	60	212	254	254	254	194	48	48	34	41	48	209	254	254	254	171	0	0	0	0	0	0
20	0	0	0	0	0	0	0	86	243	254	254	254	254	254	233	243	254	254	254	254	254	86	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	114	254	254	254	254	254	254	254	254	254	254	239	86	11	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	13	182	254	254	254	254	254	254	254	254	243	70	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	8	76	146	254	255	254	255	146	19	15	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fonte: autoria própria.

.2.2 Reconhecimento de voz

Código de importação dos dados de voz:

```
# Máximo de colunas
pd.set_option('display.max_columns', 10)

# Importação dos dados de voz
voice = pd.read_csv(DATA_PATH_RAW / 'voice.csv')

# Descrição básica dos dados
print('Dimensão:', voice.shape)
print('N duplicatas', voice.duplicated().sum())
print('N de missings:', (voice.isna()).sum().sum(),
      'Porcentagem:', round(100 * (voice.isna()).sum().sum() / (voice.shape[0] * voice.shape[1]), 2))
voice.head()
```

Fonte: autoria própria.

```
Dimensão: (3168, 21)
N duplicatas 2
N de missings: 0 Porcentagem: 0.0
```

	meanfreq	sd	median	Q25	Q75	...	mindom	maxdom	dfrange	modindx	label
0	0.059781	0.064241	0.032027	0.015071	0.090193	...	0.007812	0.007812	0.000000	0.000000	male
1	0.066009	0.067310	0.040229	0.019414	0.092666	...	0.007812	0.054688	0.046875	0.052632	male
2	0.077316	0.083829	0.036718	0.008701	0.131908	...	0.007812	0.015625	0.007812	0.046512	male
3	0.151228	0.072111	0.158011	0.096582	0.207955	...	0.007812	0.562500	0.554688	0.247119	male
4	0.135120	0.079146	0.124656	0.078720	0.206045	...	0.007812	5.484375	5.476562	0.208274	male

Fonte: autoria própria.

Distribuição do gênero dos indivíduos:

```
voice = voice.drop_duplicates()
voice['label'].value_counts()
```

Fonte: autoria própria.

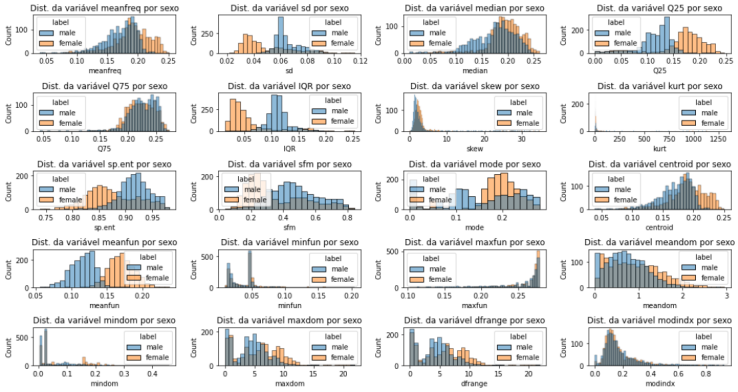
```
male      1583
female    1583
Name: label, dtype: int64
```

Fonte: autoria própria.

Distribuição dos inputs dos dados, código responsável pela imagem 29

```
fig, ax = plt.subplots(5, 4, figsize=(15, 8))

for i, col in enumerate(voice.columns[:-1]):
    sns.histplot(data=voice, x=col, hue='label', ax=ax[i//4, i%4])

    ax[i//4, i%4].set_title('Dist. da variável {} por sexo'.format(col))

plt.tight_layout()
plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

.2.3 Classificação de texto

Importação de dados dos e-mails:

```
# Máximo de coluns
pd.set_option('display.max_columns', 10)

# Importação da base de digitos
emails = pd.read_csv(DATA_PATH_RAW / 'emails.csv')

# Descrição base
emails.set_index('Email No.', inplace=True)
print('Dimensão:', emails.shape)
print('N duplicatas', emails.duplicated().sum())
print('Mín:', emails.min().min(), 'Máx:', emails.max().max())
print('N de zeros:', (emails == 0).sum().sum(),
      'Porcentagem:', round(100 * (emails == 0).sum().sum() / (emails.shape[0] * emails.shape[1]), 2))
print('N de missings:', (emails.isna()).sum().sum(),
      'Porcentagem:', round(100 * (emails.isna()).sum().sum() / (emails.shape[0] * emails.shape[1]), 2))

emails.head()
```

Fonte: autoria própria.

```
Dimensão: (5172, 3001)
N duplicatas 541
Mín: 0 Máx: 2327
N de zeros: 14645561 Porcentagem: 94.36
N de missings: 0 Porcentagem: 0.0
```

Email No.	the	to	ect	and	for	...	military	allowing	ff	dry	Prediction
Email 1	0	0	1	0	0	...	0	0	0	0	0
Email 2	8	13	24	6	6	...	0	0	1	0	0
Email 3	0	0	1	0	0	...	0	0	0	0	0
Email 4	0	5	22	0	5	...	0	0	0	0	0
Email 5	7	6	17	1	5	...	0	0	1	0	0

Fonte: autoria própria.

Distribuição dos tipos de e-mails:

```
# Removendo duplicatas
emails = emails.drop_duplicates()

# Avaliando distribuição de emails Spam e Não spam
email_pred = emails['Prediction'].value_counts().reset_index()
email_pred['index'] = ['Não-Spam', 'Spam']
email_pred['p'] = 100*email_pred['Prediction']/email_pred['Prediction'].sum()
email_pred
```

Fonte: autoria própria.

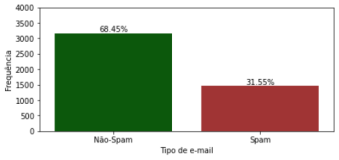
	index	Prediction	p
0	Não-Spam	3170	68.451738
1	Spam	1461	31.548262

Fonte: autoria própria.

Código responsável pelo gráfico 30

```
plt.figure(figsize=(7, 3))

# Figura
sns.barplot(x='index',y='Prediction', data=email_pred, palette={'Spam':'firebrick', 'Não-Spam':'darkgreen'})

# Escrita
for i, p in enumerate(zip(email_pred['p'], email_pred['Prediction'])):
    plt.annotate('{:.2f}%'.format(p[0]), xy=(i, p[1]), va='bottom', ha='center')

# Configuração
plt.ylim(0, 4000)
plt.ylabel('Frequência')
plt.xlabel('Tipo de e-mail')

plt.show()
```

Fonte: autoria própria.



Fonte: autoria própria.

.3 Metodologia

Os códigos abaixo necessitam que os códigos nos anexos anteriores tenham sido executados.

.3.1 Classificação de imagens

Código para a criação da tabela de frequência dos dígitos:

```
# Tabela de frequência
digits['label'].value_counts().sort_index()
```

Fonte: autoria própria.

```
0    4132
1    4684
2    4177
3    4351
4    4072
5    3795
6    4137
7    4401
8    4063
9    4188
Name: label, dtype: int64
```

Fonte: autoria própria.

Código para a execução do PCA:

```
# Método de redução de dimensionalidade PCA
pca = PCA(n_components=0.9)
X_pca = pca.fit_transform(X)

# Salvando outputs da redução (variância explicada e base gerada)
var_exp = np.sum(pca.explained_variance_ratio_)
X_pca = pd.DataFrame(X_pca)

# Variância acumulada explicada
np.cumsum(pca.explained_variance_ratio_)
```

Fonte: autoria própria.

```
array([0.09748938, 0.16909204, 0.23055107, 0.28434409, 0.33328671,
       0.37631885, 0.40908936, 0.43801039, 0.46567942, 0.48916813,
       0.51016138, 0.53075139, 0.54777693, 0.5647048 , 0.58051606,
       0.59534846, 0.60854534, 0.62137261, 0.63325237, 0.64477992,
       0.65550183, 0.66565382, 0.67530285, 0.68443131, 0.69330771,
       0.70169538, 0.70981394, 0.71758799, 0.72499434, 0.73186096,
       0.73844078, 0.74482877, 0.75082244, 0.75671157, 0.76235492,
       0.76776459, 0.77285681, 0.77773186, 0.78248756, 0.787153  ,
       0.79168253, 0.79613242, 0.80031497, 0.80429003, 0.80813545,
       0.81188465, 0.81549478, 0.81898   , 0.82234488, 0.82555226,
       0.82870693, 0.83179839, 0.83473548, 0.83760089, 0.84040849,
       0.84310467, 0.84576298, 0.84832597, 0.85086418, 0.85332596,
       0.85572312, 0.85811052, 0.86038644, 0.86260162, 0.86474096,
       0.86680229, 0.8688308 , 0.87079057, 0.87272695, 0.87461181,
       0.87647932, 0.87829602, 0.88006493, 0.88179085, 0.88345206,
       0.88508516, 0.88669117, 0.8882359 , 0.88970439, 0.89112815,
       0.89253914, 0.89394142, 0.89532977, 0.89668394, 0.89800702,
       0.89931481, 0.90061155])
```

Fonte: autoria própria.

Normalização dos dados:

```
# Normalização dos dados
scaler = preprocessing.MinMaxScaler()
X_pca = scaler.fit_transform(X_pca)
X_pca = pd.DataFrame(X_pca)
X_pca
```

Fonte: autoria própria.

	0	1	2	3	4	...	82	83	84	85	86
0	0.119352	0.246973	0.552783	0.532080	0.455475	...	0.508389	0.394086	0.440313	0.466665	0.489599
1	0.801384	0.363838	0.314767	0.601763	0.318434	...	0.429702	0.419356	0.383808	0.519651	0.566546
2	0.054326	0.386877	0.512438	0.518343	0.306548	...	0.440880	0.526999	0.473425	0.514675	0.341504
3	0.262464	0.591776	0.466821	0.739396	0.324706	...	0.282127	0.549240	0.508067	0.194113	0.280111
4	0.865533	0.333272	0.298505	0.554120	0.239025	...	0.375743	0.401654	0.354377	0.546463	0.506641
...	...	...	...	...	...	...	...	...	...	...	...
41995	0.529761	0.402026	0.347708	0.388222	0.104979	...	0.584189	0.624882	0.367948	0.519222	0.318644
41996	0.004400	0.383936	0.490128	0.505498	0.349660	...	0.376023	0.470719	0.473052	0.463464	0.357823
41997	0.407360	0.815954	0.430529	0.268257	0.281648	...	0.383017	0.362552	0.407586	0.435888	0.571450
41998	0.404800	0.509614	0.731447	0.649574	0.427508	...	0.629994	0.328498	0.348797	0.428776	0.553946
41999	0.205174	0.632729	0.510959	0.518121	0.507040	...	0.263445	0.446458	0.444291	0.463663	0.347060

Fonte: autoria própria.

Divisão dos dados entre treino e teste:

```
# Split dos dados
X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca,
                                                    y, train_size=0.75,
                                                    random_state = 0,
                                                    stratify = y)
```

Fonte: autoria própria.

.3.2 Reconhecimento de voz

Função para teste de normalidade das variáveis:

```
def normalidade(x, alpha=0.05):
    # Testes
    test1 = shapiro(x).pvalue > alpha
    test2 = jarque_bera(x).pvalue > alpha
    test3 = kstest(x, 'norm').pvalue > alpha
    return True if (test1) | (test2) | (test3) else False
```

Fonte: autoria própria.

Aplicação da função acima e seus resultados:

```
# Tipo de variável
print('N de variáveis diferentes de float:', (voice.drop(columns='label').dtypes != float).sum())
voice['label'] = voice['label'].astype(str)

# Normalidade aplicada
voice.drop(columns=['label']).apply(lambda x: normalidade(x))
```

Fonte: autoria própria.

```
N de variáveis diferentes de float: 0

meanfreq    False
sd          False
median      False
Q25         False
Q75         False
IQR         False
skew        False
kurt        False
sp.ent      False
sfm         False
mode        False
centroid    False
meanfun     False
minfun      False
maxfun      False
meandom     False
mindom      False
maxdom      False
dfrange     False
modindx     False
dtype: bool
```

Fonte: autoria própria.

Teste não-paramétrico para posição:

```
# Teste Mann_Whitney_U pelo sexo
for col in voice.columns[:-1]:
    male = voice.loc[voice['label'] == 'male', col]
    female = voice.loc[voice['label'] == 'female', col]
    test = mannwhitneyu(male, female).pvalue > 0.05

    print(col, test)
```

Fonte: autoria própria.

```
meanfreq False
sd False
median False
Q25 False
Q75 False
IQR False
skew False
kurt False
sp.ent False
sfm False
mode False
centroid False
meanfun False
minfun False
maxfun False
meandom False
mindom False
maxdom False
dfrange False
modindx True
```

Fonte: autoria própria.

Normalizando os dados:

```
# Divisão dos dados
y = voice['label'].copy()
X = voice.drop(columns=['label']).copy()

# Normalizando
scaler = preprocessing.MinMaxScaler()
X_scaler = scaler.fit_transform(X)
X_scaler = pd.DataFrame(X_scaler, columns=X.columns)
X_scaler
```

Fonte: autoria própria.

	meanfreq	sd	median	Q25	Q75	...	meandom	mindom	maxdom	dfrange	modindx
0	0.096419	0.473409	0.084125	0.060063	0.204956	...	0.000000	0.006452	0.000000	0.000000	0.000000
1	0.125828	0.505075	0.116900	0.077635	0.215683	...	0.000407	0.006452	0.002144	0.002146	0.056449
2	0.179222	0.675536	0.102873	0.034284	0.385912	...	0.000060	0.006452	0.000357	0.000358	0.049885
3	0.528261	0.554611	0.587559	0.389906	0.715802	...	0.065659	0.006452	0.025375	0.025393	0.265043
4	0.452195	0.627209	0.454272	0.317627	0.707515	...	0.238994	0.006452	0.250536	0.250715	0.223380
...	...	...	...	...	...	...	...	...	...	...	...
3161	0.436911	0.684871	0.570361	0.198513	0.686256	...	0.279703	0.006452	0.192280	0.192418	0.173674
3162	0.362946	0.731172	0.262871	0.171937	0.702595	...	0.305791	0.075269	0.167977	0.166667	0.298053
3163	0.484949	0.799042	0.690337	0.134329	0.786967	...	0.164908	0.006452	0.134024	0.134120	0.208885
3164	0.492516	0.745692	0.695311	0.175136	0.767804	...	0.265621	0.006452	0.164046	0.164163	0.333559
3165	0.595700	0.768964	0.687590	0.282629	0.901780	...	0.074312	0.006452	0.025018	0.025036	0.375386

Fonte: autoria própria.

Divisão entre base treino e teste:

```
# Split dos dados
X_train, X_test, y_train, y_test = train_test_split(X_scaler,
                                                    y, train_size=0.75,
                                                    random_state = 0,
                                                    stratify = y)
```

Fonte: autoria própria.

.3.3 Classificação de texto

Palavras mais comuns nos e-mails:

```
# Palavras mais comuns
emails.sum().sort_values(ascending=False)
```

Fonte: autoria própria.

```
e           423590
n           240301
r           227530
l           153892
c           149411
             ...
pooling         19
felipe          19
amounts         19
migration       17
hplnol          12
Length: 2867, dtype: int64
```

Fonte: autoria própria.

Identificando palavras existentes na língua inglesa:

```
# Dicionario inglês
d = enchant.Dict("en_US")

# Check e salvando quais palavras existem na lingua inglesa
columns = []
for word in tqdm(emails.columns):

    if d.check(word):
        columns.append(word)
```

Fonte: autoria própria.

Número de palavras consideradas não existentes na língua inglesa:

```
# N de palavras removidas
print('N de palavras removidas:', emails.shape[1] - len(columns))

# Salvando dataframe com palavras existentes
emails_check = emails[columns].copy()
emails_check.head()
```

Fonte: autoria própria.

```
N de palavras removidas: 598
```

Email No.	the	to	and	for	of	...	military	allowing	ff	dry	Prediction
Email 1	0	0	0	0	0	...	0	0	0	0	0
Email 2	8	13	6	6	2	...	0	0	1	0	0
Email 3	0	0	0	0	0	...	0	0	0	0	0
Email 4	0	5	0	5	1	...	0	0	0	0	0
Email 5	7	6	1	5	2	...	0	0	1	0	0

Fonte: autoria própria.

Identificação das stopwords:

```
# Lista de palavras
words = emails.columns.tolist()
words_check = emails_check.columns.tolist()

# Remoção das Stopwords
for stop in tqdm(stopwords.words('english')):
    if stop in words:
        words.remove(stop)

    if stop in words_check:
        words_check.remove(stop)

# Volume Stopwords
print('Quantidade de Stopwords na base normal:', len(emails.columns) - len(words))
print('Quantidade de Stopwords na base "Check":', len(emails_check.columns) - len(words_check))
```

Fonte: autoria própria.

```
100%|██████████| 179/179 [00:00<00:00, 48440.57it/s]
Quantidade de Stopwords na base normal: 134
Quantidade de Stopwords na base "Check": 127
```

Fonte: autoria própria.

Remoção das stopwords:

```
# Base salva
emails = emails[words].copy()
emails_check = emails_check[words_check].copy()

# Separando inputs de outputs
y = emails['Prediction'].copy()
X = emails.drop(columns=['Prediction']).copy()
X_check = emails_check.drop(columns=['Prediction']).copy()
```

Fonte: autoria própria.

Frequência relativa das palavras da base:

```
# Frequência relativa
tf_emails = X.apply(lambda x: x/ X.sum(axis=1))
tf_emails
```

Fonte: autoria própria.

Email No.	ect	hou	enron	com	gas	...	infrastructure	military	allowing	ff	dry
Email 1	0.028571	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 2	0.018127	0.020393	0.000755	0.002266	0.000755	...	0.0	0.0	0.0	0.000755	0.0
Email 3	0.014925	0.000000	0.000000	0.000000	0.029851	...	0.0	0.0	0.0	0.000000	0.0
Email 4	0.034921	0.015873	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5	0.026856	0.014218	0.000000	0.000000	0.003160	...	0.0	0.0	0.0	0.001580	0.0
...	...	...	...	...	...	...	...	...	...	...	...
Email 5168	0.006803	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5169	0.006513	0.001776	0.000592	0.001776	0.002960	...	0.0	0.0	0.0	0.000592	0.0
Email 5170	0.010417	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5171	0.002198	0.000000	0.000000	0.008791	0.000000	...	0.0	0.0	0.0	0.002198	0.0
Email 5172	0.003171	0.001268	0.000634	0.000634	0.003171	...	0.0	0.0	0.0	0.000000	0.0

Fonte: autoria própria.

Frequência relativa das palavras da base "check":

```
# Frequencia relativa de base "check"
tf_emails_check = X_check.apply(lambda x: x/ X_check.sum(axis=1))
tf_emails_check
```

Fonte: autoria própria.

Email No.	com	gas	deal	meter	please	...	infrastructure	military	allowing	ff	dry
Email 1	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 2	0.002725	0.000908	0.000000	0.000000	0.001817	...	0.0	0.0	0.0	0.000908	0.0
Email 3	0.000000	0.037037	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 4	0.000000	0.000000	0.003636	0.001818	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5	0.000000	0.003731	0.000000	0.005597	0.001866	...	0.0	0.0	0.0	0.001866	0.0
...	...	...	...	...	...	...	...	...	...	...	...
Email 5168	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5169	0.002066	0.003444	0.000000	0.000000	0.000689	...	0.0	0.0	0.0	0.000689	0.0
Email 5170	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.000000	0.0
Email 5171	0.009804	0.000000	0.002451	0.000000	0.002451	...	0.0	0.0	0.0	0.002451	0.0
Email 5172	0.000743	0.003717	0.000000	0.000000	0.000743	...	0.0	0.0	0.0	0.000000	0.0

Fonte: autoria própria.

Cálculo IDF:

```
# Calculo do idf
idf_emails = X.apply(lambda x: np.log(emails.shape[0]/(x > 0).sum()))
idf_emails
```

Fonte: autoria própria.

```
ect              0.000000
hou              0.893554
enron            1.258176
com              0.839126
gas              1.445678
                   ...
infrastructure   5.875579
military         5.607315
allowing         5.550156
ff               0.955598
dry              5.006541
Length: 2866, dtype: float64
```

Fonte: autoria própria.

Cálculo IDF na base "check":

```
# Calculo idf check
idf_emails_check = X_check.apply(lambda x: np.log(emails_check.shape[0]/(x > 0).sum()))
idf_emails_check
```

Fonte: autoria própria.

```
com              0.839126
gas              1.445678
deal             1.407904
meter            1.711899
please           0.935036
                    ...
infrastructure   5.875579
military         5.607315
allowing         5.550156
ff               0.955598
dry              5.006541
Length: 2275, dtype: float64
```

Fonte: autoria própria.

Cálculo das bases com o peso TFIDF:

```
# Multiplicando tf e idf
X_tfidf = np.multiply(tf_emails, idf_emails)
X_tfidf_check = np.multiply(tf_emails_check, idf_emails_check)
```

Fonte: autoria própria.

Divisão da base TF em treino e teste:

```
X_train, X_test, y_train, y_test = train_test_split(tf_emails,
                                                    y, train_size=0.75,
                                                    random_state = 0,
                                                    stratify = y)
```

Fonte: autoria própria.

Divisão da base "check"TF em treino e teste:

```
X_train_check, X_test_check, y_train_check, y_test_check = train_test_split(tf_emails_check,
                                                                            y, train_size=0.75,
                                                                            random_state = 0,
                                                                            stratify = y)
```

Fonte: autoria própria.

Divisão da base TFIDF em treino e teste:

```
X_train_ti, X_test_ti, y_train_ti, y_test_ti = train_test_split(X_tfidf,
                                                                y, train_size=0.75,
                                                                random_state = 0,
                                                                stratify = y)
```

Fonte: autoria própria.

Divisão da base "check"TFIDF em treino e teste:

```
X_train_ti_check, X_test_ti_check, y_train_ti_check, y_test_ti_check = train_test_split(X_tfidf_check,
                                                                                        y, train_size=0.75,
                                                                                        random_state = 0,
                                                                                        stratify = y)
```

Fonte: autoria própria.

Para testar os hiperparâmetros possíveis foi criada a função abaixo:

```
def hiper_svm(X: pd.DataFrame, y: pd.Series, c:list, gamma:list, n_jobs=-1):
    """
    Aplicação da função "GridSearchCV" para definir os melhores hiperparametros para o modelo support vector
    machine para classificação.
    """

    # Modelos
    modelo_svm = svm.SVC()

    # Parametros a serem testados
    param = {
        'C':c,
        'kernel': ['linear', 'rbf', 'sigmoid', 'poly'],
        'gamma': gamma,
    }

    # Teste
    grid = GridSearchCV(modelo_svm, param_grid=param, n_jobs=n_jobs, scoring='f1_weighted', verbose=1)
    grid.fit(X, y)

    return grid.best_params_
```

Fonte: autoria própria.

Validação cruzada entre possíveis hyperparâmetros da base TF:

```
hiper_svm(X_train, y_train, n_jobs=-1, c=[x/10 for x in list(range(1, 20))],
          gamma=['scale'])
```

Fonte: autoria própria.

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.1min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 22.0min
[Parallel(n_jobs=-1)]: Done 380 out of 380 | elapsed: 41.3min finished
{'C': 1.9, 'gamma': 'scale', 'kernel': 'poly'}
```

Fonte: autoria própria.

Validação cruzada entre possíveis hyperparâmetros da base "check"TF:

```
hiper_svm(X_train_check, y_train_check, n_jobs=-1, c=[x/10 for x in list(range(1, 20))],
          gamma=['scale'])
```

Fonte: autoria própria.

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 18.6min
[Parallel(n_jobs=-1)]: Done 380 out of 380 | elapsed: 35.9min finished
{'C': 1.9, 'gamma': 'scale', 'kernel': 'poly'}
```

Fonte: autoria própria.

Validação cruzada entre possíveis hyperparâmetros da base TFIDF:

```
hiper_svm(X_train_ti, y_train_ti, n_jobs=-1, c=[x/10 for x in list(range(1, 20))],
          gamma=['scale'])
```

Fonte: autoria própria.

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.2min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 24.1min
[Parallel(n_jobs=-1)]: Done 380 out of 380 | elapsed: 45.9min finished
{'C': 0.7, 'gamma': 'scale', 'kernel': 'sigmoid'}
```

Fonte: autoria própria.

Validação cruzada entre possíveis hyperparâmetros da base “check”TFIDF:

```
hiper_svm(X_train_ti_check, y_train_ti_check, n_jobs=-1, c=[x/10 for x in list(range(1, 20))],
          gamma=['scale'])
```

Fonte: autoria própria.

```
Fitting 5 folds for each of 76 candidates, totalling 380 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed: 19.2min
[Parallel(n_jobs=-1)]: Done 380 out of 380 | elapsed: 36.9min finished

{'C': 0.7, 'gamma': 'scale', 'kernel': 'sigmoid'}
```

Fonte: autoria própria.

.4 Resultados

Foi criada a função abaixo para execução dos modelos e avaliação dos resultados.

```
def supportvm(X_train: pd.DataFrame, X_test: pd.DataFrame,
              y_train: pd.Series, y_test: pd.Series,
              C:float, gamma:float, kernel:str):
    """
    Aplicação do modelo support vector machine e seus resultados.
    """
    # Aplicação do modelo
    start = datetime.now()
    svc = svm.SVC(C=C, gamma=gamma, kernel=kernel)
    svc.fit(X_train, y_train)
    end = datetime.now()
    print("Tempo de treino: {}".format(end - start))

    pred = svc.predict(X_test)

    # Lost function modelo
    print(classification_report(y_test, pred, zero_division=0))

    # Matrix
    matrix_confusion = pd.DataFrame(confusion_matrix(y_test, pred))

    for i in matrix_confusion.index:
        matrix_confusion.loc[i] = round(100 * matrix_confusion.loc[i] / matrix_confusion.loc[i].sum(), 2)

    # Grafico
    plt.figure(figsize=(9, 9))

    # Figuras
    ax = sns.heatmap(matrix_confusion, annot=True, vmin=0, vmax=100, linewidths=.5, fmt='g',
                     cmap=sns.color_palette("ch:start=.2,rot=-.3", as_cmap=True))

    # Configurações
    ax.set_xlabel('Valores Previstos')
    ax.set_ylabel('Valores Reais')
    ax.xaxis.set_ticklabels(list(set(y_train.unique())))
    ax.yaxis.set_ticklabels(list(set(y_train.unique())))
    plt.show()
```

Fonte: autoria própria.

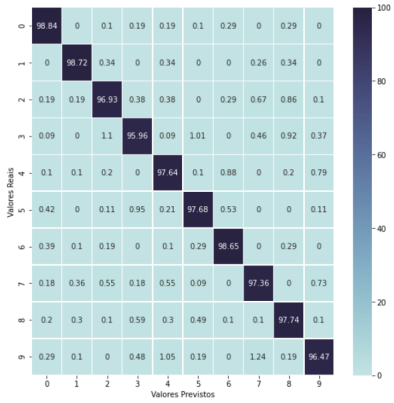
Modelo de classificação de imagens:

```
# Executando o modelo e calculando suas estatísticas
supportvm(X_train_pca, X_test_pca, y_train_pca, y_test_pca, C=1.0, gamma='scale', kernel='rbf')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:34.335301
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1033
           1       0.99      0.99      0.99      1171
           2       0.97      0.97      0.97      1044
           3       0.97      0.96      0.97      1088
           4       0.97      0.98      0.97      1018
           5       0.97      0.98      0.98       949
           6       0.98      0.99      0.98      1034
           7       0.97      0.97      0.97      1100
           8       0.97      0.98      0.97      1016
           9       0.98      0.96      0.97      1047

    accuracy                           0.98     10500
   macro avg       0.98      0.98      0.98     10500
weighted avg       0.98      0.98      0.98     10500
```

Fonte: autoria própria.



Fonte: autoria própria.

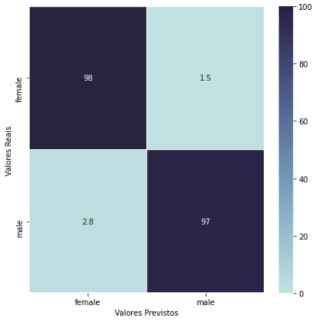
Modelo de reconhecimento de voz:

```
# Execucao do modelo
supportvm(X_train, X_test, y_train, y_test, C = 1, gamma = 'scale', kernel = 'rbf')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:00.045496
              precision    recall  f1-score   support

      female       0.97      0.98      0.98       396
        male       0.98      0.97      0.98       396

    accuracy                           0.98       792
   macro avg       0.98      0.98      0.98       792
weighted avg       0.98      0.98      0.98       792
```

Fonte: autoria própria.



Fonte: autoria própria.

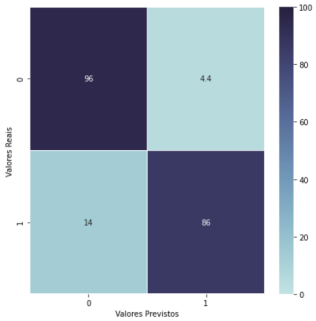
Modelo de classificação de e-mails TF:

```
# Execucao do modelo com tf
supportvm(X_train, X_test, y_train, y_test,
          C = 1.9, gamma = 'scale', kernel = 'poly')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:16.148881
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       793
           1       0.90      0.86      0.88       365

    accuracy                           0.93      1158
   macro avg       0.92      0.91      0.91      1158
weighted avg       0.93      0.93      0.93      1158
```

Fonte: autoria própria.



Fonte: autoria própria.

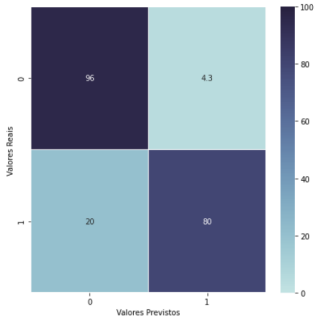
Modelo de classificação de e-mails TF "check":

```
# Execucao modelo tf_check
supportvm(X_train_check, X_test_check, y_train_check, y_test_check,
          C = 1.9, gamma = 'scale', kernel = 'poly')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:14.859576
              precision    recall  f1-score   support

           0       0.91      0.96      0.93       793
           1       0.90      0.80      0.84       365

    accuracy                           0.91      1158
   macro avg       0.90      0.88      0.89      1158
weighted avg       0.91      0.91      0.91      1158
```

Fonte: autoria própria.



Fonte: autoria própria.

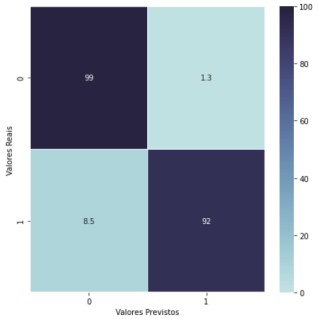
Modelo de classificação de e-mails TFIDF:

```
# Execucao tfidf
supportvm(X_train_ti, X_test_ti, y_train_ti, y_test_ti,
          C = 0.7, gamma = 'scale', kernel = 'sigmoid')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:21.663310
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       793
           1       0.97      0.92      0.94       365

    accuracy                           0.96      1158
   macro avg       0.97      0.95      0.96      1158
weighted avg       0.96      0.96      0.96      1158
```

Fonte: autoria própria.



Fonte: autoria própria.

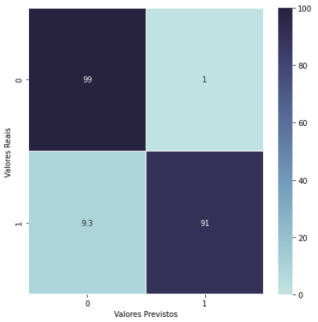
Modelo de classificação de e-mails TFIDF “check”:

```
# Execucao tfidf_check
supportvm(X_train_ti_check, X_test_ti_check, y_train_ti_check, y_test_ti_check,
          C = 0.7, gamma = 'scale', kernel = 'sigmoid')
```

Fonte: autoria própria.

```
Tempo de treino: 0:00:36.492790
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       793
           1       0.98      0.91      0.94       365

    accuracy                           0.96      1158
   macro avg       0.97      0.95      0.96      1158
weighted avg       0.96      0.96      0.96      1158
```

Fonte: autoria própria.



Fonte: autoria própria.