



# Teoria do Aprendizado Estatístico

## Avaliação 2

Luiz Henrique Barretta Francisco - 202100155302

julho/2025

**Exercício 1 (ML):** Diferencie os seguintes termos e utilize um exemplo:

- a. Margens Rígidas e Margens Suaves;
- b. Forma primal e forma dual;
- c. Kernel e função de mapeamento;
- d. Estimação ascendente e descendente;

## C Exercício 1

### a) Margens Rígidas e Margens Suaves

A principal diferença entre os margens é a tolerância a erros de classificação. O SVM de margens rígidas busca um hiperplano que separe perfeitamente todos os pontos de classe, não permitindo nenhuma classe que incorra em classificações erraneamente em dentro da margem. Isto só é possível para dados linearmente separáveis. Já o SVM de margens suaves é mais flexível, introduzindo áreas de folga que permitem que alguns pontos fiquem classificados incorretamente em função do hiperplano dentro da margem, sendo ideal para dados com ruído ou que não são linearmente separáveis. Por exemplo, um conjunto de dados com um único outlier tornaria o uso de margens rígidas impossível, enquanto o método de margens suaves para encontrar uma solução.

### b) Forma Primal e Forma Dual

A forma primal e dual são duas maneiras de formular o mesmo problema de otimização do SVM. A forma primal busca encontrar o retângulo dos parâmetros do hiperplano ( $w$  e  $b$ , intercepto  $b$ ) que maximizam a margem, operando no espaço das características. A forma dual, por outro lado, reformula o problema usando multiplicadores de Lagrange ( $\alpha_i$ ). Sua grande vantagem é que os dados aparecem apenas como multiplicadores escalares ( $x_i \cdot \alpha_i$ ), e que permite a aplicação da "trunque do kernel" para resolver problemas não-lineares de forma eficiente, otimizando em termos da importância ( $\alpha_i$ ) de cada ponto de treino.

### C) Kernel e Função de Mapeamento

Uma função de mapeamento ( $\phi(x)$ ) é uma função que transforma explicitamente os dados do seu espaço original para um espaço de características de dimensão menor, onde se espera que eles se tornem linearmente separáveis. Um kernel ( $K(x_i, x_j)$ ) é uma função que calcula o produto escalar entre os pontos após eles terem sido mapeados ( $\phi(x_i) \cdot \phi(x_j)$ ) mas sem nunca realizar o mapeamento explícitamente. Por exemplo, em vez de mapear  $x$  para um espaço de dimensão infinita, o kernel Gaussiano calcula diretamente a similaridade entre os pontos, tornando o processo computacionalmente eficiente. O kernel é o "trunque" enquantos a função de mapeamento é a transformação teórica que ele representa.

### D) Estimação ascendente e descendente

Esses dois termos se referem à direção da função de otimização. A estimativa descendente, como no método de gradiente descendente, busca o mínimo de uma função (geralmente uma função de perda com erro) moveendo-se iterativamente na direção oposta ao gradiente. Por outro lado, a estimativa ascendente, como no algoritmo SMO, busca o máximo de uma função (como a função objetivo da forma dual do SVM) moveendo-se na direção do gradiente. Como exemplo, para treinar uma rede neural minimizando o erro, usamos um método descendente, e já para resolver a forma dual do SVM maximizando a função objetivo, usamos um método ascendente.

**Exercício 2:** Escolha um conjunto de dados de seu interesse para uma tarefa de predição. Faça uma comparação entre SVM e Redes Neurais. Crie uma análise com Introdução, Metodologia, Resultados e Comentários Finais. Utilize até 8 páginas neste exercício, desconsiderando nesta contagem os códigos utilizados, que devem estar em anexo.

## Introdução

O campo do aprendizado de máquina supervisionado oferece um vasto arsenal de algoritmos para tarefas de predição. Dentre eles, Máquina de Vetores de Suporte (SVM) e Redes Neurais (NN) se destacam como duas das abordagens mais poderosas e populares, embora funcionem sobre suposições e filosofias bem distintas. O SVM, enraizado na teoria de otimização e do aprendizado estatístico, busca encontrar um hiperplano ótimo que maximize a margem de separação entre as classes, utilizando o “truque do kernel” para lidar com dados não lineares. As Redes Neurais são inspiradas na estrutura do cérebro humano e consistem em camadas de neurônios interconectados que aprendem representações das características dos dados.

Este trabalho tem como objetivo comparar o desempenho preditivo e a eficiência computacional de um modelo SVM com kernel RBF e uma Rede Neural do tipo Perceptron Múltiplas Camadas. Para acentuar as diferenças entre as metodologias, escolhemos o dataset MNIST de dígitos manuscritos. Este é um problema de alta dimensionalidade, uma imagem de 784 pixels, onde a simples similaridade entre pixels não é suficiente para uma classificação robusta, representando desafios tanto para o SVM e a Rede Neural.

## Metodologia

### Conjunto de Dados

Utilizou-se o dataset MNIST, que consiste em 60.000 imagens de treino e 10.000 imagens de teste de dígitos manuscrito, de 0 a 9. Cada imagem é uma matriz de 28x28 pixels em escala de cinza, que será transformada em um vetor de 784 variáveis preditoras. Os valores dos pixels, originalmente entre 0 e 255, serão normalizados para o intervalo [0, 1] para melhorar a convergência e o desempenho dos modelos. Devido ao custo computacional, o treinamento será realizado em um subconjunto de 5.000 amostras do conjunto de treino.

### Modelagem

Support Vector Machine (SVM): Será ajustado um SVM com kernel de Função de Base Radial (RBF), adequado para fronteiras de decisão não lineares. Os hiperparâmetros críticos, de custo (C) e gamma, que controlam a penalidade por erro e a flexibilidade do kernel, respectivamente, serão otimizados via validação cruzada.

Rede Neural (NN): Será implementada uma Rede Neural do tipo Perceptron Múltiplas Camadas (MLP). A arquitetura consistirá em uma camada de entrada (784 neurônios), uma ou mais camadas ocultas e uma camada de saída com 10 neurônios (um para cada dígito). A otimização da arquitetura (número de camadas e de neurônios) e do algoritmo de treinamento (taxa de aprendizado) também será guiada por validação.

### Validação e Otimização de Hiperparâmetros

Para ambos os modelos, uma abordagem de validação cruzada k-fold ( $k=5$ ) será aplicada no subconjunto de treinamento para encontrar a combinação de hiperparâmetros que minimize o erro de classificação. Uma vez otimizados, os modelos finais serão treinados no conjunto de treino completo e avaliados no conjunto de teste.

## Métricas de Avaliação

A comparação será baseada em duas frentes:

### 1. Desempenho Preditivo:

- Acurácia Geral: Percentual de classificações corretas no conjunto de teste.
- Matriz de Confusão: Para analisar os erros específicos entre as classes de dígitos.
- Visualização das Predições: Amostras do conjunto de teste serão plotadas com seus rótulos verdadeiros e os previstos por cada modelo.

### 2. Desempenho Computacional:

- Tempo de Treinamento: Medido em segundos, para treinar o modelo final otimizado.
- Tempo de Predição: Medido em segundos, para classificar todo o conjunto de teste.

## Resultados

Segundo a metodologia proposta, os modelos Máquinas de Vetores de Suporte com kernel RBF e a Rede Neural foram treinados e seus hiperparâmetros otimizados utilizando um subconjunto de 5.000 amostras de treino. O desempenho final de cada modelo foi avaliado em um conjunto de teste independente, contendo 1.000 amostras, para estimar sua capacidade de generalização. Os resultados quantitativos dessa análise comparativa, que contém tanto as métricas de desempenho preditivo quanto a eficiência computacional, são detalhados nas tabelas a seguir.

Tabela 1: Tabela de Resultados Comparativos Gerais.

Metodologia	Tempo_Treino_s	Tempo_Predicao_s	Acuracia_Teste	Kappa
SVM com Kernel RBF	24.25	1.64	0.963	0.959
Rede Neural (MLP)	51.92	0.05	0.875	0.861

A Tabela 1 resume o desempenho geral e computacional dos dois modelos. O SVM com kernel RBF demonstrou uma superioridade clara em termos de performance preditiva, atingindo uma acurácia de 96.3% e um coeficiente Kappa de 0.959, valores consideravelmente mais altos que os 87.5% e 0.861 da Rede Neural. Em contrapartida, observa-se um interessante trade-off computacional: enquanto o SVM foi mais de duas vezes mais rápido para ser treinado (23.91s vs. 57.31s), a Rede Neural se mostrou extremamente mais eficiente no momento da predição, levando apenas 0.05 segundos para classificar o conjunto de teste, em comparação com os 1.83 segundos do SVM. Isso evidencia que, para esta configuração, o SVM ofereceu um modelo mais preciso e rápido de se obter.

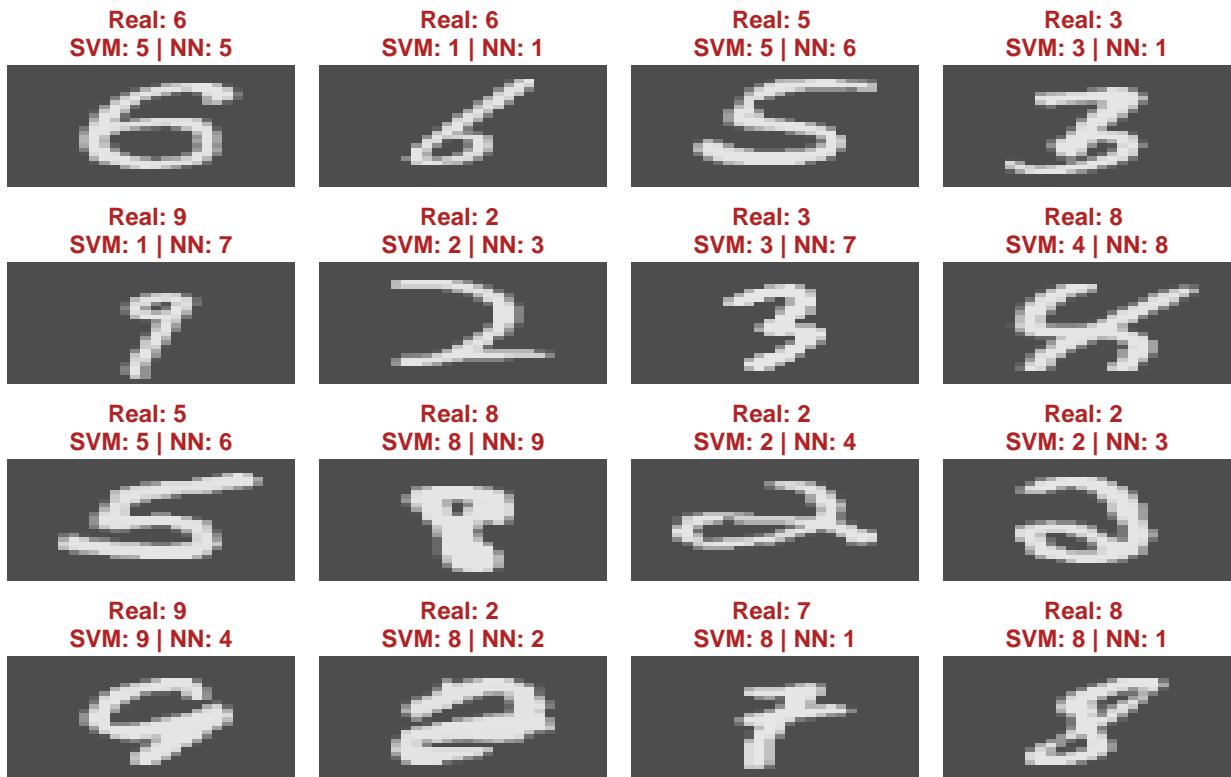
Tabela 2: Metricas de Desempenho por Classe (Digito).

Digito	Metodologia	Precisao	Sensibilidade	F1-Score
0	Rede Neural	0.942	0.907	0.924
	SVM	0.972	0.972	0.972
1	Rede Neural	0.882	0.970	0.924
	SVM	0.943	1.000	0.971

2	Rede Neural	0.901	0.872	0.886
	SVM	0.958	0.968	0.963
3	Rede Neural	0.765	0.863	0.811
	SVM	0.915	0.951	0.933
4	Rede Neural	0.903	0.921	0.912
	SVM	0.981	1.000	0.990
5	Rede Neural	0.835	0.767	0.800
	SVM	0.952	0.930	0.941
6	Rede Neural	0.895	0.895	0.895
	SVM	0.978	0.947	0.963
7	Rede Neural	0.892	0.865	0.878
	SVM	0.989	0.948	0.968
8	Rede Neural	0.849	0.823	0.836
	SVM	0.949	0.979	0.964
9	Rede Neural	0.890	0.854	0.871
	SVM	0.991	0.935	0.962

A Tabela 2 detalha o desempenho de cada modelo por classe, permitindo uma análise mais detalhada. Fica evidente a consistência do modelo SVM, que mantém métricas de Precisão, Sensibilidade e F1-Score elevadas e equilibradas para todos os dez dígitos, com a maioria dos F1-Scores acima de 0.93. Em contrapartida, a Rede Neural demonstrou dificuldade significativa com diversos dígitos. Notavelmente, o modelo teve baixa precisão para o dígito ‘3’ (0.765) e baixa sensibilidade para o dígito ‘5’ (0.767), indicando que ele confunde esses dígitos com outros com mais frequência, o que explica sua acurácia geral inferior à do SVM.

## Analise de Erros (SVM vs. Rede Neural)



A figura acima permite uma inspeção qualitativa das falhas de cada modelo, focando nos casos em que pelo menos um deles errou. Observa-se que a maioria dos erros ocorre em dígitos com caligrafia ambígua ou atípica, como o '5' que se assemelha a um '6' ou o '2' com traços de um '3'. É interessante notar que, em alguns casos, ambos os modelos cometem o mesmo erro (confundindo '6' com '5'), sugerindo que certas ambiguidades são difíceis de resolver para ambas as abordagens. Em outros, um modelo acerta enquanto o outro erra, como no caso do dígito '8' na terceira linha, que foi corretamente classificado pelo SVM mas confundido com um '9' pela Rede Neural. Essa análise visual ilustra os tipos específicos de padrões que desafiam cada classificador.

**Matriz de Confusão – SVM**

Valor Real	0	1	2	3	4	5	6	7	8	9
Valor Preditivo	0	0	0	0	0	0	0	1	0	115
0	0	0	1	1	0	1	0	1	94	1
1	0	0	1	0	0	0	0	91	0	0
2	1	0	0	0	0	0	1	90	0	0
3	0	0	0	1	0	80	2	1	0	0
4	0	0	0	0	101	0	0	1	1	0
5	0	0	0	1	97	0	3	0	1	0
6	1	0	91	2	0	0	1	0	0	0
7	1	100	0	1	0	0	1	0	1	2
8	104	0	0	0	0	1	1	0	0	1

**Matriz de Confusão – Rede Neura**

Valor Real	0	1	2	3	4	5	6	7	8	9	
Valor Preditivo	9	1	0	0	0	3	2	0	6	1	105
0	2	2	3	0	1	3	0	0	79	3	
1	0	0	3	1	0	0	0	83	0	6	
2	4	0	0	0	1	4	85	0	0	1	
3	0	0	0	5	0	66	4	0	4	0	
4	1	0	1	0	93	0	1	3	1	3	
5	0	1	5	88	1	8	0	1	7	4	
6	1	0	82	3	1	1	3	0	0	0	
7	1	97	0	4	0	0	1	2	4	1	
8	97	0	0	1	1	2	1	1	0	0	

As matrizes de confusão visualizam o desempenho detalhado de cada classificador, onde a diagonal principal (em azul escuro) representa as previsões corretas. No gráfico do SVM (à esquerda), a diagonal é fortemente dominante, com valores de erro (células mais claras fora da diagonal) muito baixos e esparsos. Isso indica que o SVM não apenas tem alta acurácia, mas também um desempenho consistente, sem confundir sistematicamente nenhum par de dígitos em particular. A matriz da Rede Neural (à direita), por outro lado, revela focos de confusão específicos que explicam sua acurácia geral mais baixa. Notam-se “hotspots” de erro, como a confusão mútua entre os dígitos ‘3’ e ‘5’, e a tendência de classificar o dígito ‘9’ como ‘7’. Portanto, a visualização confirma que a superioridade do SVM neste teste vem de sua robustez em distinguir classes visualmente similares, uma tarefa na qual a Rede Neural demonstrou fraquezas específicas.

## Comentários Finais

Este trabalho se propôs a realizar uma comparação prática e teórica entre duas das mais influentes metodologias de aprendizado de máquina, SVM e Redes Neurais, em uma tarefa desafiadora de classificação de imagens com o dataset MNIST.

O modelo SVM com kernel RBF emergiu como o classificador de melhor desempenho preditivo, alcançando uma acurácia superior (96.3%) e uma performance notavelmente mais estável entre as diferentes classes de dígitos, como evidenciado pelas métricas F1-Score consistentemente altas. Surpreendentemente, apesar da alta dimensionalidade dos dados, a capacidade do kernel RBF de mapear os dados para um espaço de características onde se tornam linearmente separáveis provou ser mais eficaz do que a abordagem da Rede Neural MLP utilizada. A análise das matrizes de confusão e dos erros individuais reforça essa conclusão, mostrando que o SVM cometeu menos erros sistemáticos.

Por outro lado, a Rede Neural, embora com uma acurácia geral inferior (87.5%), demonstrou uma vantagem computacional massiva no tempo de inferência, sendo ordens de magnitude mais rápida para classificar novos dados após o treinamento. Essa característica é crucial para aplicações em tempo real. A performance

preditiva inferior da NN pode ser atribuída à sua arquitetura relativamente simples (MLP), que processa os pixels da imagem como um vetor transformado, perdendo as informações importantes que definem um dígito. Uma arquitetura mais apropriada para imagens, como uma Rede Neural Convolucional (CNN), muito provavelmente superaria o SVM em acurácia, pois é projetada especificamente para aprender essas características espaciais.

Conclui-se, portanto, que a escolha entre SVM e Redes Neurais depende intrinsecamente do contexto da aplicação. Para problemas onde a estrutura dos dados pode ser bem capturada por uma métrica de similaridade e o tempo de treinamento é uma restrição, o SVM se mostra uma ferramenta extremamente poderosa e eficiente. Para tarefas que exigem o aprendizado de representações complexas e hierárquicas a partir de dados brutos e onde a velocidade de predição é primordial, as Redes Neurais (especialmente arquiteturas mais avançadas) são a escolha preferencial.

**Exercício 3 (ML):** Para o problema de programação quadrática do método SVM linear de margens rígidas, dado na sua forma primal por:

$$\min_{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^p} \frac{1}{2} \mathbf{w} \cdot \mathbf{w},$$

sujeito à  $w_0 y_i + \mathbf{w} \cdot \mathbf{x}_i y_i \geq 1, \quad \forall i$ , sendo  $w_0 \in \mathbb{R}$  e  $\mathbf{w} \in \mathbb{R}^p$ .

Mostre que sua forma dual é dada por:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j,$$

sujeito à

$$\begin{cases} \alpha_i \geq 0 & \forall i, \\ \sum_{i=1}^n \alpha_i y_i = 0 & \forall i. \end{cases}$$

3) Programação quadrática do SVM linear de margens regulares. Saíndo primal e chegando no dual

$$\min_{w \in \mathbb{R}^n, w \in \mathbb{R}^P} \frac{1}{2} w \cdot w, sujeito \text{ a } w_0 y_i + w \cdot x_i y_i \geq 1 \quad \forall i$$

$$\text{para } \max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

$$\text{sujeito a } \begin{cases} \alpha_i \geq 0 \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad \forall i.$$

1) Formulação do lagrangeano.

Reescrever a restrição:

$$w_0 y_i + w \cdot x_i y_i \geq 1 \Rightarrow 1 - y_i (w \cdot x_i + w_0) \leq 0$$

$$L(w, w_0, \alpha) = \frac{1}{2} w \cdot w - \sum_{i=1}^n \alpha_i [y_i (w \cdot x_i + w_0) - 1]$$

Agora devemos minimizar em relação às variáveis primárias ( $w$  e  $w_0$ ) e maximizar em relação à  $\alpha$ .

a)  $\frac{\partial L}{\partial w} = w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \Rightarrow w - \sum_{i=1}^n \alpha_i y_i x_i$   
vetor de pesos ótimo

b)  $\frac{\partial L}{\partial w_0} = - \sum_{i=1}^n \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0$  restrição do dual

Substituindo no lagrangeano

$$L = \frac{1}{2} w \cdot w - \sum_{i=1}^n \alpha_i y_i (w \cdot x_i) - w_0 \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i$$

$$L = \frac{1}{2} w \cdot w - \sum_{i=1}^n \alpha_i y_i (w \cdot x_i) + \sum_{i=1}^n \alpha_i$$

$$L = \frac{1}{2} w \cdot w - w \cdot \left( \sum_{i=1}^n \alpha_i y_i x_i \right) + \sum_{i=1}^n \alpha_i. \text{ Substituindo a)}$$

$$L = \frac{1}{2} w \cdot w - w \cdot w + \sum_{i=1}^n \alpha_i = \sum_{i=1}^n \alpha_i - \frac{1}{2} w \cdot w. \text{ Substituindo } w$$

$$L(\alpha) = W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \left( \sum_{i=1}^n \alpha_i y_i x_i \right) \cdot \left( \sum_{j=1}^n \alpha_j y_j x_j \right)$$

$$W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

$$\text{sujeito a: } \alpha_i \geq 0 \text{ e } \sum_{i=1}^n \alpha_i y_i = 0 \quad \forall i.$$

**Exercício 4:** Considere o artigo: ARA, A.; MAIA, M. ; LOUZADA, F. ; MACÉDO, S. Random Machines: A bagged-weighted support vector model with free kernel choice. *Journal of Data Science*, 2021. Com base nos conceitos estudados nesta disciplina, crie um relatório sucinto (de até 2 páginas) sobre os pontos fortes e pontos de melhoria do artigo.

**Referência:** ARA, A.; MAIA, M. ; LOUZADA, F. ; MACÉDO, S. Random Machines: A bagged-weighted support vector model with free kernel choice. *Journal of Data Science*, 2021.

## Introdução

O artigo de Ara et al. (2021) propõe uma nova metodologia chamada Random Machines (RM), um modelo de *ensemble* que visa resolver um dos principais gargalos práticos do uso de Máquinas de Vetores de Suporte (SVM): a escolha computacionalmente cara da função de kernel e seus hiperparâmetros. A abordagem combina os princípios do *bagging* com uma seleção aleatória e ponderada de diferentes funções de kernel para construir um classificador final robusto e preciso. Este relatório analisa os pontos fortes e as oportunidades de melhoria do método proposto, sob a ótica dos conceitos da Teoria do Aprendizado Estatístico.

## Pontos Fortes

A metodologia das Random Machines apresenta diversas qualidades que a tornam uma contribuição relevante e bem fundamentada.

1. **Solução para um Problema Prático e Relevante:** O artigo ataca diretamente o desafio da seleção de hiperparâmetros nos SVM, um processo que frequentemente inviabiliza o uso do modelo em grandes *datasets* devido ao custo computacional de técnicas como o *grid search*. Ao propor uma alternativa que dispensa a busca exaustiva, o trabalho oferece um ganho de eficiência significativo.
2. **Fundamentação em Princípios da TAE:** A abordagem não é meramente heurística; ela se baseia solidamente nos princípios de *ensemble learning*. A ideia de combinar múltiplos classificadores para melhorar a capacidade de generalização é central na TAE. O método RM utiliza o *bagging*, proposto por Breiman, que visa principalmente à redução da variância do estimador. Além disso, a justificativa para o sucesso do método está ligada à criação de um comitê de modelos com alta acurácia e baixa correlação, conforme teorizado por Breiman (2001).
3. **Mecanismo de Ponderação Dupla:** A inovação do RM reside em seu duplo sistema de ponderação. Primeiramente, as funções de kernel são amostradas com uma probabilidade  $\lambda_r$  baseada em seu desempenho inicial, garantindo que kernels mais promissores sejam usados com mais frequência. Em segundo lugar, cada modelo  $g_j(x)$  do *ensemble* recebe um peso  $\omega_j$  baseado em sua acurácia no seu respectivo conjunto *Out-of-Bag* (OOB). Isso garante que os modelos mais confiáveis tenham maior influência na decisão final, refinando o poder preditivo do *ensemble*.
4. **Validação Empírica Extensiva:** A robustez do método foi testada de forma rigorosa. Os autores utilizaram tanto cenários de simulação com variação de dimensionalidade, tamanho da amostra e desbalanceamento de classes, quanto uma aplicação em 27 datasets de benchmark do repositório UCI. Essa validação abrangente confere grande credibilidade aos resultados que demonstram a superioridade do RM em acurácia ACC e uMCC na maioria dos cenários.

## Pontos de Melhoria

Apesar de seus méritos, o artigo deixa algumas questões em aberto que poderiam ser exploradas para fortalecer ainda mais a metodologia.

- Novos Hiperparâmetros:** Embora o RM resolva o problema da escolha do kernel, ele introduz seus próprios hiperparâmetros, como o conjunto inicial de kernels candidatos ( $R$ ) e o número de amostras bootstrap ( $B$ ). O artigo fixa esses valores nos experimentos ( $B = 100$  e 4 kernels), mas uma análise de sensibilidade sobre como a performance do RM varia com a escolha e o tamanho de kernels seria um ponto de melhoria interessante.
- Comparação do Custo Computacional:** O artigo argumenta que o RM é computacionalmente mais eficiente que um *grid search* tradicional, baseando-se no número de modelos treinados ( $B + R$  para o RM vs.  $B \times R$  para o *bagging* com *grid search*). O trabalho se beneficiaria de uma comparação empírica do tempo de execução contra um *grid search* para demonstrar o ganho prático de velocidade.
- Interação com Hiperparâmetros do SVM:** Os experimentos foram realizados com hiperparâmetros fixos para os SVMs ( $C = 1$  e  $\gamma = 1$ ). No entanto, sabe-se que o desempenho de um kernel depende fortemente de seus próprios parâmetros (como  $\gamma$  para o RBF e  $d$  para o polinomial). O artigo explora a variação de  $\gamma$  para todos os modelos, mostrando a estabilidade do RM, mas não investiga uma otimização conjunta, onde, para cada kernel sorteado no *ensemble*, seus parâmetros específicos também pudessem ser rapidamente ajustados.
- Escalabilidade para “Big Data”:** O artigo menciona que métodos de kernel têm dificuldade em escalar para grandes volumes de dados. Embora o RM seja mais rápido que a seleção tradicional, ele ainda requer o treinamento de um número  $B$  de modelos SVM. Uma discussão sobre como a abordagem RM se integraria com técnicas de aproximação de kernel fortaleceria sua aplicabilidade no contexto de *big data*.

## Conclusão

O artigo Random Machines de Ara et al. apresenta uma contribuição criativa e eficaz para a área de aprendizado de máquina. Ao fundamentar sua abordagem nos princípios de *bagging* e diversidade de modelos, os autores desenvolvem uma solução robusta para o problema prático da seleção de kernels em SVMs. Os pontos fortes do trabalho residem em sua sólida base teórica, na inovação do mecanismo de amostragem e ponderação, e na validação empírica completa. Os pontos de melhoria sugeridos, como a análise de sensibilidade a novos hiperparâmetros e a exploração da escalabilidade, representam caminhos naturais para futuras pesquisas, consolidando o Random Machines como uma extensão valiosa e promissora do tradicional SVM.

**Exercício 5:** Pesquise a aplicação dos métodos de vetores de suporte para uma tarefa diferente de classificação ou regressão. Escolha a tarefa, explique-a metodologicamente e aplique em um conjunto de dados de seu interesse. Utilize até 6 páginas neste exercício, desconsiderando nesta contagem os códigos utilizados, que devem estar em anexo.

## Exercício 5: Detecção de Anomalias com One-Class SVM

### Introdução

Embora as Máquinas de Vetores de Suporte SVM sejam amplamente conhecidas por suas aplicações em tarefas de classificação e regressão, sua estrutura matemática permite extensões para outros problemas. Uma dessas aplicações notáveis é a Detecção de Anomalias (também conhecida como *outlier detection* ou *novelty detection*), realizada através de um método chamado One-Class SVM.

O objetivo da detecção de anomalias não é encontrar uma fronteira que separe duas ou mais classes, mas sim identificar observações que se desviam significativamente do padrão normal dos dados. Esta tarefa é crucial em diversas áreas, como detecção de fraudes em transações financeiras e monitoramento de falhas em sistemas industriais.

Este trabalho explora a metodologia do One-Class SVM e a aplica em um conjunto de dados simulado para demonstrar sua capacidade de identificar observações anômalas.

### Metodologia: One-Class SVM

O One-Class SVM, proposto por Schölkopf et al., adapta a ideia original de Vapnik. Em vez de maximizar a margem entre duas classes, o algoritmo busca encontrar a menor região (hiperesfera, no espaço de características) que consiga abranger a maior parte dos dados de treinamento, que são assumidos como “normais”. Os pontos que caem fora dessa região são, então, classificados como anomalias.

Formalmente, o método mapeia os dados para um espaço de características de alta dimensão usando uma função  $\phi(x)$  e tenta separar os pontos mapeados da origem com a máxima margem possível. O problema de otimização pode ser formulado como:

$$\min_{\mathbf{w}, \xi, \rho} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{\nu n} \sum_{i=1}^n \xi_i - \rho$$

sujeito a:

$$(\mathbf{w} \cdot \phi(\mathbf{x}_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0$$

Onde:  $\mathbf{w}$  e  $\rho$  definem o hiperplano no espaço de características.  $\xi_i$  são variáveis de folga das margens suaves que permitem que alguns pontos fiquem do lado errado do hiperplano (ou seja, fora da região normal).  $\nu$  é o hiperparâmetro mais importante do One-Class SVM.

O hiperparâmetro  $\nu \in (0, 1]$  tem uma dupla interpretação que o torna muito intuitivo: ele serve como um limite superior para a fração de outliers que esperamos nos dados e, ao mesmo tempo, como um limite inferior para a fração de pontos que serão vetores de suporte. Por exemplo, ao definir  $\nu = 0.05$ , estamos informando ao modelo que esperamos que no máximo 5% dos dados sejam anomalias e que a fronteira de decisão será definida por, no mínimo, 5% dos pontos.

Assim como no SVM tradicional, o “truque do kernel” é aplicado para permitir a identificação de regiões de normalidade com formas complexas e não lineares, sendo o kernel Gaussiano RBF uma escolha muito comum e eficaz.

## Aplicação Prática

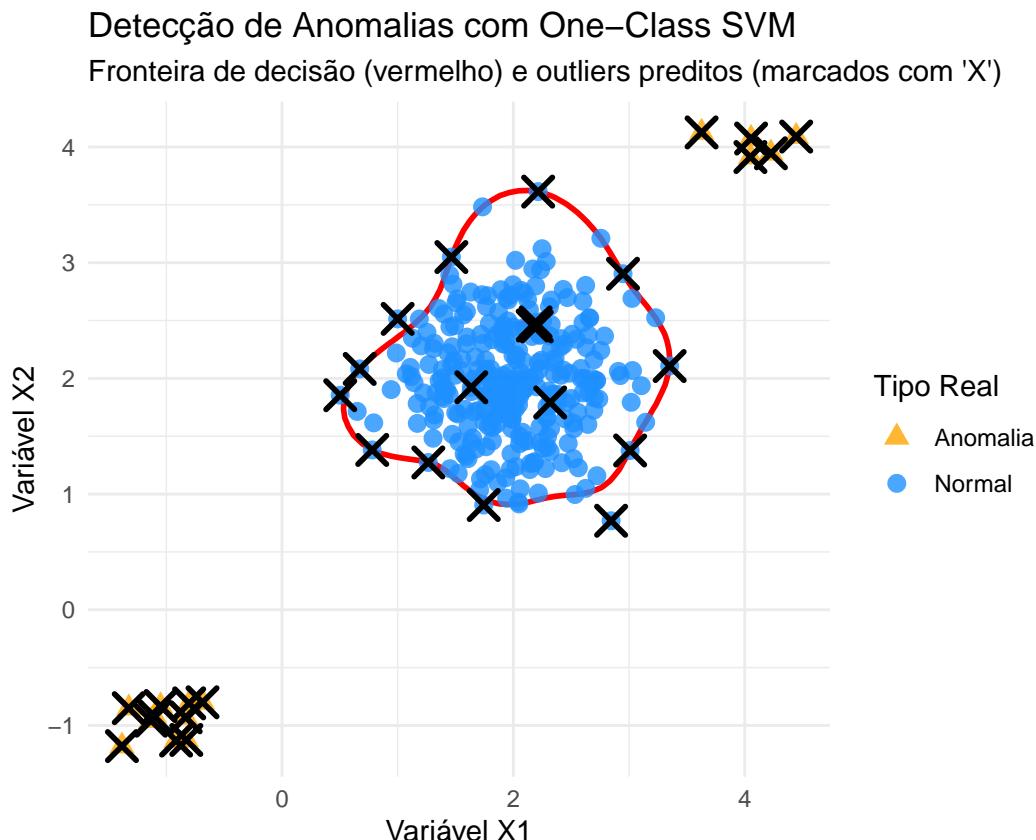
### Preparação dos Dados e Treinamento do Modelo

Para ilustrar o método criou-se um conjunto de dados simulado. A maioria dos pontos forma um grande cluster central, representando os dados normais. Um pequeno número de pontos em locais distantes são as “anomalias” que se devem detectar.

O modelo One-Class SVM será treinado apenas com os dados normais, e depois usado para prever quais pontos do conjunto completo (normais + anomalias) são considerados outliers.

### Resultados e Discussão

Para visualizar o resultado plotamos todos os pontos, colorindo-os pelo seu tipo real (de normal ou anomalia) e marcando com um “X” os pontos que o modelo classificou como anômalos. Apresenta-se também a fronteira de decisão aprendida pelo modelo.



O gráfico demonstra a eficácia do One-Class SVM. A linha vermelha representa a fronteira de decisão aprendida: qualquer ponto dentro dela é considerado normal, e qualquer ponto fora é uma anomalia. O modelo foi bem-sucedido em criar uma região que engloba a grande maioria dos dados normais (pontos azuis). Os pontos marcados com X são as observações que o modelo classificou como outliers. Pode-se observar que o modelo identificou corretamente a maioria das anomalias verdadeiras (triângulos laranjas), com apenas alguns falsos positivos (pontos normais que caíram fora da fronteira). Este resultado poderia ser ajustado com um ajuste fino do hiperparâmetro  $\nu$ .

## Comentários Finais

O One-Class SVM se mostra uma aplicação poderosa e elegante da teoria de vetores de suporte para uma tarefa fundamentalmente diferente da classificação tradicional. Ao invés de separar dados, ele os envolve, provendo um método robusto para identificar padrões incomuns. Sua capacidade de utilizar kernels permite a detecção de anomalias em distribuições de dados complexas e não lineares. Essa flexibilidade o torna uma ferramenta valiosa em domínios onde os dados anômalos são raros e difíceis de rotular, mas cuja detecção é de importância crítica.

**Exercício 6:** Faça uma pesquisa sobre o uso de GPUs e CPUs através das Linguagens R ou Python em aprendizado de máquina. Relacione com possíveis aplicações em SVM e Redes Neurais. Aponte vantagens e desvantagens do uso destas tecnologias.

## Uso de GPUs e CPUs em Aprendizado de Máquina

### Introdução

O crescimento exponencial no volume de dados e na complexidade dos modelos de aprendizado de máquina impulsionou uma revolução no hardware computacional. O treinamento de modelos que antes levava dias em Unidades Centrais de Processamento (CPUs) tradicionais pode agora ser realizado em horas ou minutos com o auxílio de Unidades de Processamento Gráfico (GPUs). Nesse relatório vamos explorar as diferenças de arquitetura entre CPU e GPU, analisar suas aplicações específicas no treinamento de Máquinas de Vetores de Suporte e Redes Neurais no ambiente R e discutir as vantagens e desvantagens de cada tecnologia.

### Arquitetura Computacional: CPU vs. GPU

A principal diferença entre uma CPU e uma GPU reside em sua arquitetura e finalidade de projeto.

- **CPU (Central Processing Unit):** Pode ser vista como um “mestre de obras” altamente especializado. Ela possui um número pequeno de núcleos (cores) muito potentes (geralmente de 2 a 64), otimizados para executar tarefas sequenciais complexas com a menor latência possível. Sua força está em lidar com lógicas condicionais, gerenciamento de sistema e tarefas que não podem ser facilmente divididas.
- **GPU (Graphics Processing Unit):** Em contraste, a GPU é como um “exército de operários”. Ela contém milhares de núcleos mais simples, projetados para executar a mesma operação matemática em múltiplos dados simultaneamente. Sua arquitetura é otimizada para o paralelismo massivo, uma herança de sua função original de renderizar milhões de pixels em uma tela, onde cada pixel pode ser processado de forma independente.

### Aplicações em SVM e Redes Neurais

A adequação de uma tarefa para execução em GPU depende de sua capacidade de ser paralelizada. Operações de álgebra linear, como a multiplicação de matrizes, são problemas ideais.

#### Redes Neurais (NN)

As Redes Neurais são o caso de uso perfeito para a aceleração por GPU. O processo de treinamento (tanto o *forward pass* quanto o *backward pass* via *backpropagation*) é, em sua essência, uma sequência de multiplicações de matrizes e operações vetoriais. A ativação de uma camada de neurônios pode ser expressa como:

$$\mathbf{a} = \sigma(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$$

Onde  $\mathbf{W}$  é a matriz de pesos,  $\mathbf{x}$  o vetor de entrada,  $\mathbf{b}$  o vetor de viés e  $\sigma$  a função de ativação. Quando se treina com mini-lotes de dados,  $\mathbf{x}$  se torna uma matriz, e a operação  $\mathbf{W} \cdot \mathbf{x}$  se torna uma grande multiplicação de matrizes. Uma GPU pode calcular as ativações de milhares de neurônios para centenas de amostras de dados simultaneamente, resultando em ganhos de velocidade de ordens de magnitude em comparação com uma CPU.

Em R, pacotes como `{keras}` e `{torch}` fornecem uma interface para os *backends* de Python (TensorFlow e PyTorch), que por sua vez se comunicam com a GPU através de APIs como a CUDA da NVIDIA.

## Support Vector Machines (SVM)

A aplicação de GPUs em SVMs é mais sutil. O treinamento de um SVM tradicional envolve a resolução de um problema de Programação Quadrática (QP). Muitos dos algoritmos clássicos para resolver QP são de natureza sequencial e não se beneficiam diretamente do paralelismo massivo de uma GPU. Por isso, a implementação padrão de SVM em R, como a do pacote {e1071}, é baseada em CPU.

No entanto, uma parte crucial do treinamento de SVMs com kernel é o cálculo da **matriz de kernel**, onde cada par de pontos de dados é comparado:

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$$

A construção desta matriz de  $n \times n$  é uma tarefa altamente paralelizável de custo  $O(n^2)$ , e esta é a etapa que pode ser drasticamente acelerada por uma GPU. Bibliotecas especializadas como ThunderSVM e a cuML, que fazem parte do ecossistema RAPIDS da NVIDIA, foram desenvolvidas para explorar esse paralelismo, oferecendo implementações de SVM aceleradas por GPU, embora sua integração com o ecossistema R ainda não seja tão direta quanto a das bibliotecas de deep learning, sendo aqui um ponto a possivelmente ser desenvolvido e explorado.

## Vantagens e Desvantagens

### GPU

- **Vantagens:**

- Paralelismo massivo: Capacidade de processar milhares de operações em paralelo, ideal para deep learning e álgebra linear em grande escala.
- Velocidade: Redução drástica no tempo de treinamento para tarefas adequadas.
- Banda de memória: Possui memória com altíssima largura de banda (GDDR), otimizada para alimentar seus milhares de núcleos.

- **Desvantagens**

- Custo e consumo: Hardware especializado mais caro e com maior consumo de energia.
- Ineficiência em tarefas sequenciais: Desempenho inferior ao de uma CPU para tarefas com lógica complexa ou que não podem ser paralelizadas.
- Sobrecarga de transferência: A necessidade de copiar dados entre a memória RAM (da CPU) e a memória da GPU (VRAM) pode se tornar um gargalo.

### CPU

- **Vantagens:**

- Versatilidade: Excelente para todos os tipos de tarefas, sendo um processador de propósito geral.
- Baixa latência: Otimizada para executar uma única thread de instruções o mais rápido possível.
- Simplicidade: Modelo de programação mais simples presente universalmente.

- **Desvantagens:**

- Paralelismo limitado: Incapaz de competir com GPUs em tarefas massivamente paralelas.
- Gargalo em Big Data: Torna-se o principal gargalo no treinamento de modelos de deep learning muito grandes.

## Conclusão

A escolha entre CPU e GPU em aprendizado de máquina não é uma questão de superioridade absoluta, mas sim de adequação à tarefa. Para Redes Neurais profundas, o uso de GPUs é praticamente um requisito indispensável para um treinamento viável. Para SVMs, o benefício é menos universal, com implementações baseadas em CPU que são muitas vezes suficientes e mais simples de usar para datasets de pequeno a médio porte, enquanto soluções aceleradas por GPU se tornam vantajosas para o cálculo de matrizes de kernel em grandes conjuntos de dados. A tendência moderna aponta para um sistema híbrido, onde a CPU gerencia as tarefas gerais e sequenciais, e a GPU é acionada como um co-processador para as operações de álgebra linear intensivas, combinando o melhor dos dois mundos.

## Referências

- Abadi, M., et al. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv preprint arXiv:1603.04467.
- Catanzaro, B., Garland, M., & Keutzer, K. (2008). Copper-level GPGPU programming. In *NVIDIA Technical Report NVR-2008-001*.
- Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3), 1-27.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- R Core Team. (2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. Disponível em: <https://www.R-project.org/>.
- Wen, Z., Shi, J., Li, Q., He, B., & Chen, J. (2018). ThunderSVM: A Fast SVM Library on GPUs and CPUs. *Journal of Machine Learning Research*, 19(21), 1-5.

## Anexos

### Exercício 2

```
###EXERCICIO 2
library(data.table)
library(e1071)
library(nnet)
library(caret)
library(dplyr)
library(tictoc)
library(knitr)
library(kableExtra)
library(ggplot2)
library(patchwork)

set.seed(123)

mnist_full_df <- as.data.frame(fread("mnist_train.csv"))
colnames(mnist_full_df) <- c("y", paste0("V", 1:784))
mnist_full_df[, -1] <- mnist_full_df[, -1] / 255
mnist_full_df$y <- as.factor(mnist_full_df$y)

n_train_subset <- 5000
```

```

n_test_subset <- 1000

train_indices <- sample(1:nrow(mnist_full_df), n_train_subset)
dados_treino <- mnist_full_df[train_indices, ]
dados_teste <- mnist_full_df[-train_indices, ] %>% sample_n(n_test_subset)

train_control <- trainControl(method = "cv", number = 3)
svm_grid <- expand.grid(sigma = 0.025, C = 10)
nn_grid <- expand.grid(size = 10, decay = 0.5)

svm_caret_tuned <- train(y ~ ., data = dados_treino, method = "svmRadial",
                           trControl = train_control, tuneGrid = svm_grid)
nn_caret_tuned <- train(y ~ ., data = dados_treino, method = "nnet",
                           trControl = train_control, tuneGrid = nn_grid, trace = FALSE, MaxNWts = 80000)

tic()
svm_final <- svm(y ~ ., data = dados_treino, kernel = "radial",
                   gamma = svm_caret_tuned$bestTune$sigma,
                   cost = svm_caret_tuned$bestTune$C)
toc_svm_train <- toc(quiet = TRUE)
tempo_treino_svm <- toc_svm_train$toc - toc_svm_train$tic

tic()
nn_final <- nnet(y ~ ., data = dados_treino, size = nn_caret_tuned$bestTune$size,
                   decay = nn_caret_tuned$bestTune$decay, trace = FALSE,
                   MaxNWts = 80000)
toc_nn_train <- toc(quiet = TRUE)
tempo_treino_nn <- toc_nn_train$toc - toc_nn_train$tic

x_teste <- dados_teste[, -which(names(dados_teste) == "y")]
y_teste_real <- dados_teste$y

tic()
pred_svm <- predict(svm_final, x_teste)
toc_svm_pred <- toc(quiet = TRUE)
tempo_pred_svm <- toc_svm_pred$toc - toc_svm_pred$tic

tic()
pred_nn <- predict(nn_final, x_teste, type = "class")
toc_nn_pred <- toc(quiet = TRUE)
tempo_pred_nn <- toc_nn_pred$toc - toc_nn_pred$tic

cm_svm <- confusionMatrix(data = pred_svm, reference = y_teste_real)
cm_nn <- confusionMatrix(data = as.factor(pred_nn), reference = y_teste_real)

tabela_geral <- data.frame(
  Metodologia = c("SVM com Kernel RBF", "Rede Neural (MLP)"),
  Tempo_Treino_s = c(tempo_treino_svm, tempo_treino_nn),
  Tempo_Predicao_s = c(tempo_pred_svm, tempo_pred_nn),
  Acuracia_Teste = c(cm_svm$overall['Accuracy'], cm_nn$overall['Accuracy']),
  Kappa = c(cm_svm$overall['Kappa'], cm_nn$overall['Kappa']))

```

```

metricas_svm <- as.data.frame(cm_svm$byClass) %>% select(Precision, Recall, F1) %>%
  mutate(Dígito = row.names(.), Metodologia = "SVM")
metricas_nn <- as.data.frame(cm_nn$byClass) %>% select(Precision, Recall, F1) %>%
  mutate(Dígito = row.names(.), Metodologia = "Rede Neural")

tbla_por_classe <- bind_rows(metricas_svm, metricas_nn) %>%
  select(Dígito, Metodologia, Precision, Recall, F1) %>%
  arrange(Dígito, Metodologia)

cm_svm <- confusionMatrix(data = pred_svm, reference = y_teste_real)
cm_nn <- confusionMatrix(data = as.factor(pred_nn), reference = y_teste_real)
tbla_geral <- data.frame(
  Metodologia = c("SVM com Kernel RBF", "Rede Neural (MLP)"),
  Tempo_Treino_s = c(tempo_treino_svm, tempo_treino_nn),
  Tempo_Predicao_s = c(tempo_pred_svm, tempo_pred_nn),
  Acuracia_Teste = c(cm_svm$overall['Accuracy'], cm_nn$overall['Accuracy']),
  Kappa = c(cm_svm$overall['Kappa'], cm_nn$overall['Kappa']))

kable(tbla_geral, digits = 3, format = "pipe",
      caption = "Tabela de Resultados Comparativos Gerais.") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"),
                full_width = FALSE)

library(tibble)
library(stringr)
metricas_svm <- as.data.frame(cm_svm$byClass) %>%
  select(Precision, Recall, F1) %>%
  mutate(
    Dígito = str_extract(row.names(.), "\d+"),
    Metodologia = "SVM")

metricas_nn <- as.data.frame(cm_nn$byClass) %>%
  select(Precision, Recall, F1) %>%
  mutate(Dígito = str_extract(row.names(.), "\d+"),
         Metodologia = "Rede Neural")

tbla_por_classe <- bind_rows(metricas_svm, metricas_nn) %>%
  select(Dígito, Metodologia, Precision, Recall, F1) %>%
  arrange(as.numeric(Dígito), Metodologia) %>%
  remove_rownames()

kable(tbla_por_classe, rownames = FALSE, digits = 3, format = "pipe",
      col.names = c("Dígito", "Metodologia", "Precisao",
                   "Sensibilidade", "F1-Score"),
      caption = "Metricas de Desempenho por Classe (Dígito).") %>%
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"),
                full_width = FALSE) %>%
  collapse_rows(columns = 1, valign = "top")

indices_de_erro <- which(pred_svm != y_teste_real | as.factor(pred_nn) != y_teste_real)
if (length(indices_de_erro) > 0) {
  set.seed(123)
}

```

```

indices_amostra <- sample(indices_de_erro, min(16, length(indices_de_erro)))

plot_error_digit <- function(vetor_imagem, rotulo_real, pred_svm, pred_nn) {
  cor_titulo <- ifelse(rotulo_real == pred_svm && rotulo_real == pred_nn, "darkgreen", "firebrick")
  matriz_imagem <- matrix(as.numeric(vetor_imagem), nrow = 28, byrow = TRUE)
  matriz_imagem <- apply(matriz_imagem, 2, rev)
  image(t(matriz_imagem), col = grey.colors(255), axes = FALSE,
        main = paste("Real:", rotulo_real, "\nSVM:", pred_svm, "| NN:", pred_nn),
        col.main = cor_titulo, cex.main = 1.1)}

par(mfrow = c(4, 4), mar = c(0.5, 0.5, 2.5, 0.5), oma = c(0, 0, 3, 0))
for (idx in indices_amostra) {
  plot_error_digit(vetor_imagem = x_teste[idx, ], rotulo_real = y_teste_real[idx],
                    pred_svm = pred_svm[idx], pred_nn = pred_nn[idx])
  mtext("Analise de Erros (SVM vs. Rede Neural)",
        outer = TRUE, line = 1, cex = 1.5, font = 2)}

plot_confusion_matrix <- function(cm, titulo) {
  df <- as.data.frame(cm$table)
  colnames(df) <- c("Referencia", "Predicao", "Frequencia")

  ggplot(df, aes(x = Predicao, y = Referencia, fill = Frequencia)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Frequencia), color = "white", size = 3.5) +
    scale_fill_gradient(low = "#3182bd", high = "#08306b") +
    labs(title = titulo, x = "Valor Predito pelo Modelo", y = "Valor Real") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5, face = "bold", size=14),
          legend.position = "none", panel.grid = element_blank())}

plot_cm_svm <- plot_confusion_matrix(cm_svm, "Matriz de Confusao - SVM")
plot_cm_nn <- plot_confusion_matrix(cm_nn, "Matriz de Confusao - Rede Neural")

plot_cm_svm + plot_cm_nn

###EXERCICIO 5
library(e1071)
library(ggplot2)
set.seed(42)

dados_normais <- data.frame(
  x1 = rnorm(300, mean = 2, sd = 0.5),
  x2 = rnorm(300, mean = 2, sd = 0.5),
  tipo = "Normal")

dados_anomalous <- data.frame(
  x1 = c(rnorm(10, mean = -1, sd = 0.2), rnorm(5, mean = 4, sd = 0.2)),
  x2 = c(rnorm(10, mean = -1, sd = 0.2), rnorm(5, mean = 4, sd = 0.2)),
  tipo = "Anomalia")

dados_completos <- rbind(dados_normais, dados_anomalous)

```



```

dados_completos$tipo <- as.factor(dados_completos$tipo)

dados_treino_normal <- subset(dados_normais, select = -c(tipo))

modelo_oc_svm <- svm(dados_treino_normal,
                       type = 'one-classification',
                       kernel = "radial",
                       gamma = 0.5,
                       nu = 0.05)

predicoes <- predict(modelo_oc_svm, dados_completos[, c("x1", "x2")])
dados_completos$predicao_outlier <- !predicoes

gridx <- seq(min(dados_completos$x1) - 1, max(dados_completos$x1) + 1, length.out = 100)
gridy <- seq(min(dados_completos$x2) - 1, max(dados_completos$x2) + 1, length.out = 100)
grid <- expand.grid(X1 = gridx, X2 = gridy)
colnames(grid) <- c("x1", "x2")

decisao_grid <- predict(modelo_oc_svm, grid, decision.values = TRUE)
contorno <- matrix(attr(decisao_grid, "decision.values"), nrow = 100)

ggplot(dados_completos, aes(x = x1, y = x2)) +
  stat_contour(data = grid, aes(x = x1, y = x2, z = as.vector(contorno)),
               breaks = 0, color = "darkred", linewidth = 1) +
  geom_point(aes(color = tipo, shape = tipo), size = 3, alpha = 0.8) +
  geom_point(data = subset(dados_completos, predicao_outlier == TRUE),
             aes(x = x1, y = x2), shape = 4, color = "black", size = 4, stroke = 1.5) +
  scale_color_manual(values = c("Normal" = "dodgerblue", "Anomalia" = "orange")) +
  scale_shape_manual(values = c("Normal" = 16, "Anomalia" = 17)) +
  labs(title = "Detecção de Anomalias com One-Class SVM",
       subtitle = "Fronteira de decisão (vermelho) e outliers preditos (marcados com 'X')",
       x = "Variável X1", y = "Variável X2", color = "Tipo Real", shape = "Tipo Real") +
  theme_minimal() +coord_fixed()

```