Rotulagem de Imagens para Projetos de Visão Computacional

Autores:
Cecília Hélen Nunes Câmara - 500593
Diego Rabelo de Sá - 539664
Louis Ian Silva dos Santos - 402525

Abstract

Este projeto visa aprimorar a rotulagem de imagens em projetos de visão computacional, lidando com desafios de qualidade, como desfoque e iluminação inadequada. Utilizando técnicas de aprendizado de máquina, são desenvolvidos cinco modelos capazes de classificar automaticamente a qualidade das imagens. O processo envolve a organização das imagens, extração de características relevantes, construção do conjunto de dados, treinamento dos modelos e avaliação de seu desempenho. Esse trabalho contribui para a eficácia dos modelos de visão computacional, filtrando imagens de baixa qualidade.

Introdução

Motivação

Durante a coleta de dados para tarefas de visão computacional, frequentemente lidamos com imagens de baixa qualidade a ponto de dificultar a rotulação delas e, possivelmente, prejudicar o desempenho do modelo se forem utilizadas durante o treinamento. Portanto, nosso projeto visa a criação de um modelo capaz de detectar automaticamente imagens de baixa qualidade e descartá-las de um conjunto de imagens arbitrário.

É importante ressaltar que, dentro do escopo da área de visão computacional, nosso objetivo é classificar imagens que são **fotografias** mundanas, sem teor artístico.

Modelos

Avaliamos cinco algoritmos diferentes, todos de **classificação binária** e implementados na biblioteca *Scikit-learn* da linguagem de programação *Python*, sendo estes:

Modelo	Classe do *Scikit-learn*
Regressão Logística	SGDClassifier
Máquinas de Vetor Suporte	SVC
K-Nearest Neighbours	KNeighboursClassifier
Random Forest	RandomForestClassifier
Naive Bayes Gaussiano	GaussianNB

Fundamentação Teórica

Extração de atributos das imagens

Intuitivamente, os fatores que caracterizam uma fotografia (de tamanho em *pixels* arbitrário) de baixa qualidade, são a iluminação ruim (por exemplo, ela ser muito escura) e um alto grau de desfoque.

Portanto, foi necessário extrair essas informações de forma a estruturar os dados gerados. Trivialmente, a informação relacionada ao brilho da imagem pode ser extraída pelos valores de cada *pixel* da fotografia. Assim, usamos as medidas descritivas de **média** e **desvio padrão** para cada canal de uma imagem.

Antes de calcular esses valores, como nosso problema não leva em consideração o tamanho da imagem, redimensionamos-a para um valor fixo de $512x512$ para que não haja algum viés relacionado a seu tamanho original.

Para o cálculo da média, temos uma média aritmética simples:

$$\mu_C = \frac{1}{512^2} \sum_{p \in P_C} p, \forall C \in \{R, G, B\}$$

Aqui, μ_C é a média dos valores dos *pixels* de um canal de cor C, e P_C é o conjunto de *pixels* desse canal.

O mesmo vale para o desvio padrão:

$$\sigma_C = \sqrt{\frac{\sum_{p \in P_C} (p - \mu_C)^2}{512^2}}, \forall C \in \{R, G, B\}$$

Assim, totalizamos seis atributos iniciais relacionados às cores de uma imagem:

$$[\mu_R, \mu_G, \mu_B, \sigma_R, \sigma_G, \sigma_B]$$

Quanto à informação de desfoque, optamos por técnicas de extração de bordas em imagens. Especificamente, utilizamos o operador ***Prewitt***, um filtro de convolução que destaca transições abruptas de intensidade na imagem. Essas transições indicam mudanças significativas na textura ou nos limites dos objetos presentes na imagem. Essas informações são valiosas para avaliar o grau de desfoque e contribuem para o processo de detecção de imagens de baixa qualidade:

$$G_h(I) = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * I$$

$$G_v(I) = \begin{bmatrix} +1 & 0 & -1 \\ +1 & 0 & -1 \\ +1 & 0 & -1 \end{bmatrix} * I$$

Em que $G_a, \forall a \in \{h, v\}$ representa a matriz resultante da operação para cada eixo (horizontal ou vertical) da imagem, I a imagem em **preto e branco**, e $*$ o operador de convolução.

Para transformar a imagem em preto e branco, utilizamos uma média aritmética para cada pixel:

$$p_{xy} = \frac{\sum_{C \in \{R,G,B\}} p_{Cxy}}{3}, \forall x, y \in \{1, 2, ...512\}$$

Dessa forma, como não é relevante o sentido das bordas (esquerda ou direita, acima ou abaixo), utilizamos o valor absoluto da matriz resultante, denotada por G_a, para obter a matriz de valores absolutos P_a.

$$P_a = |G_a(I)|, \forall a \in \{h, v\}$$

Assim, utilizamos novamente as medidas descritivas de média e desvio padrão sobre essa matriz, e assim, temos mais quatro atributos:

$$[\mu_R, \mu_G, \mu_B, \sigma_R, \sigma_G, \sigma_B, \mu_h, \mu_v, \sigma_h, \sigma_v]$$

Totalizando o total de atributos gerados durante o processamento sobre um imagem arbitrária.

Coleta de dados e rotulação

O conjunto de dados foi construídos de uma mistura de DataSets públicos do *site* Kaggle, sendo um para cada tipo de imagem (boa qualidade, desfocadas e baixa luminosidade):

Classe da imagem	Link
Boa qualidade	Flickr Image dataset
Desfocada	Blur dataset
Baixa iluminação	Exclusively-Dark-Image-Dataset

Para a construção das imagens de **Boa qualidade** utilizamos todas as 700 imagens desfocadas dos *Blur dataset*, sendo 350 borradas e 350 com desfoque de movimento.

Com o objetivo de compor um conjunto de dados balanceado, selecionamos aleatoriamente 700 imagens do *Flickr dataset* para compor a classe de **Má qualidade**.

Além disso, seria interessante fornecer imagens com baixa iluminação para o treinamento dos modelos. Por isso, selecionamos 700 imagens do *Exclusively-Dark-Image-Dataset* para obter 350 de **Boa qualidade** e 350 de **Má qualidade**, que, por fim, foram rotuladas de acordo.

É importante observar que nem toda imagem com baixa iluminação é uma imagem de baixa qualidade.

Conjunto de dados gerado

O conjunto de dados em formato *CSV* pode ser encontrado no *GitHub* do projeto. As imagens originais foram salvas na nuvem, por questões de armazenamento.

O tamanho do conjunto de dados é de 2100 instâncias, perfeitamente balanceadas para cada classe binária, e com 10 atributos cada, sem dados faltantes.

Metodologia

Selecionamos cinco algoritmos distintos de aprendizado de classificação binária para uso no nosso projeto.

Cada algoritmo segue o padrão de *Pipelines* do *Scikit-learn*, que proporciona uma sequência de transformações de dados, como normalização, até o modelo final.

Para cada algoritmo, utilizamos *Grid Search Cross-Validation* de 5 *folds* para a escolha de hiperparâmetros, dentro de um *KFold* de 10 *folds* para avaliação de cada modelo treinado. Ambos com a *random seed* especificada arbitrariamente em 42 para manter a reprodutibilidade dos treinamentos.

Com cada modelo treinado em sua respectiva partição, utilizamos os índices de acurácia global, acurácia por classe, precisão, revocação e o escore F1 para avaliá-los. Registramos também, para o melhor e pior modelo de cada algoritmo, as curvas *Precision-Recall* e *ROC* (no caso modelos probabilísticos), além da matriz de confusão.

Para cada uma das métricas geradas, calculamos a média e o desvio padrão e geramos gráficos do tipo *BoxPlot* (consulte sessão abaixo).

Após a geração dos gráficos, re-treinamos cada modelo utilizando novamente a técnica de *Grid Search Cross-Validation* para escolher os hiperparâmetros ideais para o nosso conjunto de dados. Em seguida, o modelo é treinado utilizando esses hiperparâmetros com todo o conjunto de dados.

Experimentos

Dados

Realizamos uma **análise** e um **pré-processamento** dos dados coletados para identificar comportamentos dos atributos extraídos. Utilizamos o método ***describe*** da biblioteca Pandas para analisar os intervalos de valores das médias e desvios padrões obtidos, obtendo os seguintes resultados:

Intervalo de Valores - Média e Desvio Padrão

Atributo	Intervalo Boa	Intervalo Má
Rmean	$13.929 \leq 219.476$	$0.407 \leq 220.176$
Gmean	$10.871 \leq 214.544$	$0.543 \leq 208.023$
Bmean	$2.673 \leq 223.166$	$0.714 \leq 202.883$
Rstd	$19.064 \leq 106.034$	$4.529 \leq 97.059$
Gstd	$16.364 \leq 106.656$	$4.409 \leq 97.059$
Bstd	$5.237 \leq 106.440$	$2.473 \leq 98.168$
Hmean	$1.867 \leq 24.639$	$0.429 \leq 23.859$
Vmean	$1.654 \leq 30.877$	$0.396 \leq 21.381$
Hstd	$2.927 \leq 36.528$	$0.530 \leq 38.580$
Vstd	$3.285 \leq 39.951$	$0.502 \leq 36.470$

As imagens de boa qualidade apresentaram médias e desvios padrões maiores, em comparação com as de má qualidade. Essa tendência observada se manteve quando realizamos a separação em subclasses (*boa qualidade, escuras boas, escuras ruins, movimento, desfoque*), servindo como uma classificação inicial mais assertiva para determinar se uma imagem é de boa ou má qualidade.

Analisamos a presença de **outliers** com o uso do boxplot e seu impacto no projeto. Mostrou-se que as imagens de má

qualidade apresentaram uma maior quantidade de outliers em comparação com as de boa qualidade (Figura 1). No entanto, decidimos não modificá-los ou excluí-los, levando em consideração o tamanho da base de dados e a quantidade insignificante de outliers, o que não resultou em viés no modelo.

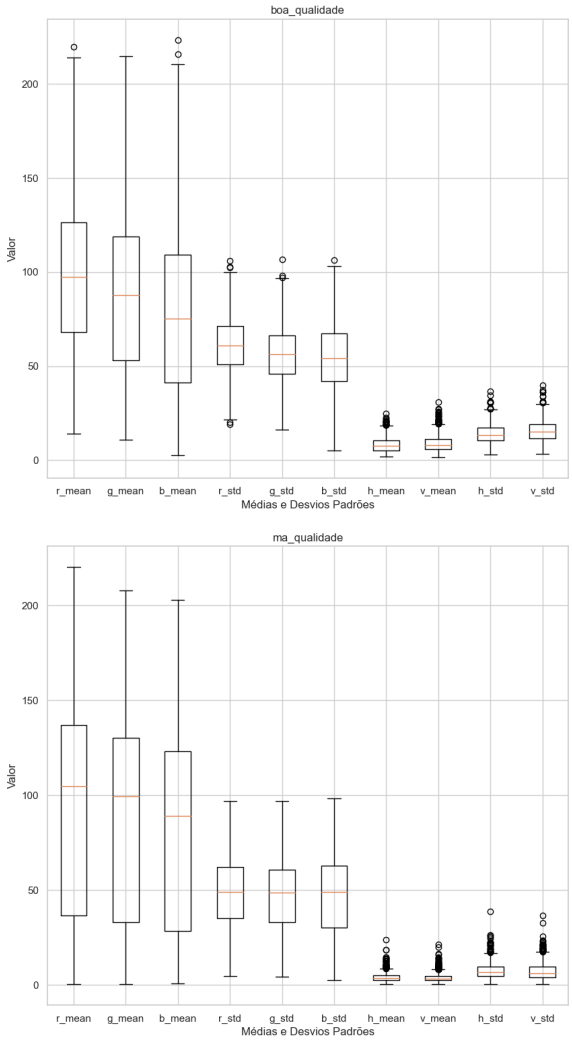

Figure 1: Boxplot das médias e desvios padrões

Desempenho dos models

Após o treinamento dos modelos propostos, obtemos os seguintes resultados:

Média das Métricas

Modelo	Acurácia	Precisão	Revocação	F1-Score
RegLog	0.87	0.88	0.86	0.86
SVM	0.88	0.88	0.88	0.88
KNN	0.86	0.85	0.86	0.86
RF	0.87	0.87	0.87	0.87
GNB	0.79	0.83	0.72	0.77

Desvio Padrão das Métricas

Modelo	Acurácia	Precisão	Revocação	F1-Score
RegLog	0.017	0.046	0.073	0.024
SVM	0.02	0.036	0.021	0.02
KNN	0.024	0.031	0.032	0.021
RF	0.022	0.033	0.018	0.019
GNB	0.018	0.03	0.03	0.02

Aqui podemos ver que, com exceção do Naive Bayes Gaussiano que demonstrou uma performance substancialmente pior do que os demais, o desempenho dos modelos é comparativamente inferior ao algoritmo do modelo escolhido, com o SVM tendo resultados levemente melhores que o Random-Forest, seguido pela regressão logística e o KNN.

Nas figuras 2 à 6 estão as métricas detalhadas por *fold* de cada modelo e boxplots dessas métricas (médias em verde):

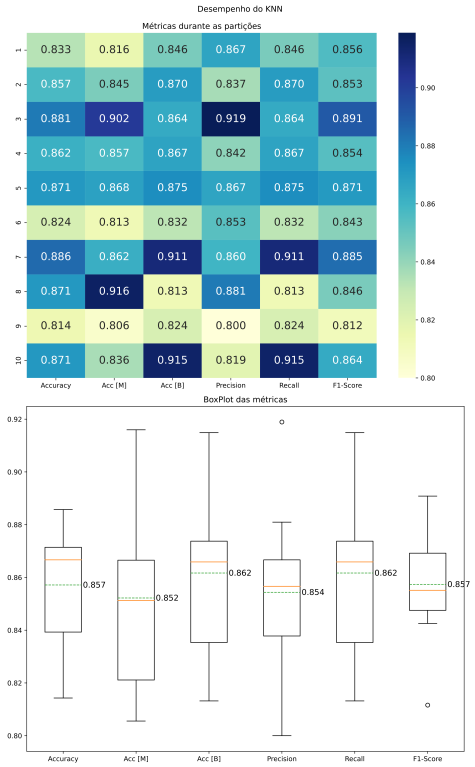

Figure 2: Métricas do KNN

Podemos fazer ainda algumas observações curiosas a partir das acurácias por classe. Primeiro notamos que a performance do *Naive Bayes Gaussiano* para detectar imagens de má qualidade (**0.860**) não está tão significativamente atrás da performance dos outros modelos (em média **0.870**), e que na realidade o que impactou sua acurácia global foi sua baixa acurácia na detecção de imagens de boa qualidade (**0.726** contra uma média de **0.865** dos demais modelos). Nota-se também que o melhor modelo na detecção de imagens de baixa qualidade foi a regressão logística, pois possui a melhor média da acurácia da classe de **Má Qualidade**, apesar de fazê-lo com uma diferença mínima: $<$ **0.002** em relação

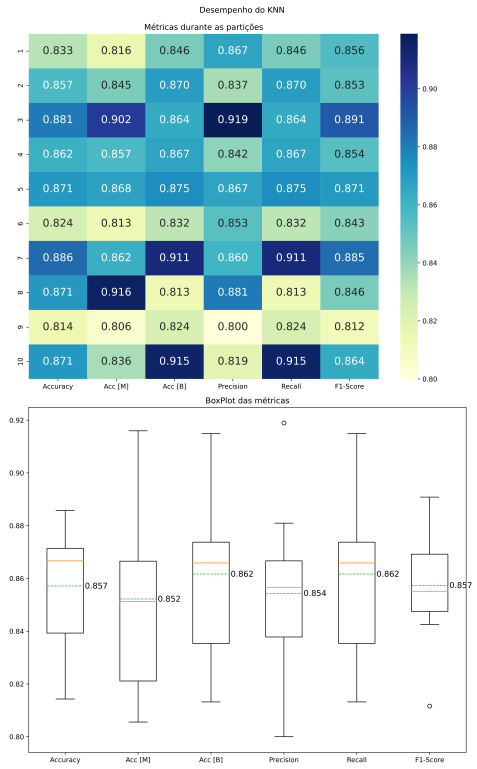

Figure 3: Métricas da Regressão Logística

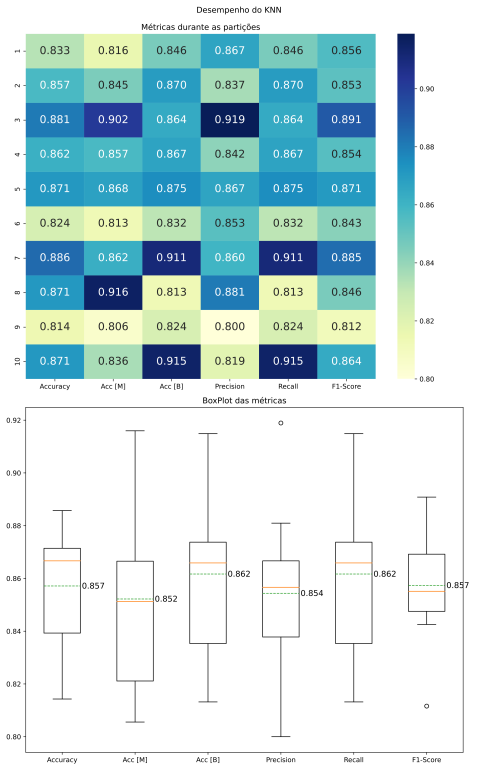

Figure 4: Métricas do Random Forest

aos demais modelos. Já no caso da detecção de imagens de **Boa Qualidade**, o SVM demonstrou a melhor performance, apesar de novamente a marge sobre o segundo melhor modelo, que neste caso foi o *Random-Forest*, ser $< \mathbf{0.004}$.

Nas figuras 8 à 12 estão as curvas *ROC* (com exceção do *KNN*), Revocação e Matriz de confusão dos.

Eis os tamanhos finais do modelos, medidos em ***kilobytes***, após retreinamento com o *dataset* completo,:

Tamanho dos modelos (em KB)

Modelo	Tamanho
Naive Bayes Gaussiano	1.85
Regressão Logística	15.2
Máquinas de Vetor Suporte	207
K-Nearest Neighbours	378
Random Forest	833

Conclusão

Ao utilizar o modelo, observamos a ocorrência de dois tipos de erros recorrentes, ambos relacionados a imagens de baixa iluminação com presença de luz vermelha. Ao analisar o *dataset* original, notamos que a maioria das imagens de baixa iluminação apresentava tons avermelhados devido à iluminação das lâmpadas de rua. Essa característica induziu o modelos a classificar erroneamente imagens com baixa iluminação, mas sem a presença da luz vermelha, como sempre sendo de baixa qualidade, o que nem sempre é verdade. Da mesma forma, o modelo também reconheceu erroneamente imagens com luz vermelha presente como sendo de boa qualidade, mesmo quando isso não era o caso.

Diante disso, um desafio a ser enfrentado em projetos futuros seria o fornecimento de mais dados para o treinamento de um modelo relacionado, bem como o equilíbrio desses dados. Vale ressaltar que, devido à indisponibilidade de um *dataset* pronto e público para o nosso projeto, foi necessário combinar três *datasets* distintos. Essas melhorias ajudariam a aprimorar a capacidade do modelos em lidar com imagens de baixa iluminação com presença de luz vermelha, proporcionando resultados mais precisos e confiáveis.

Apesar de não apresentar a melhor performance técnica final, a *regressão logística* se destaca como a melhor escolha para o modelo final do projeto devido ao seu equilíbrio entre simplicidade, alta acurácia média e eficiência computacional. Esse modelo é especialmente adequado para aplicações que visam detectar imagens de baixa qualidade. Por outro lado, o *SMV* se mostra mais apropriado para aplicações em que a detecção de imagens de boa qualidade é mais crítica.

Em situações em que a acurácia de ambas as classes é igualmente importante, pode haver oportunidades de explorar ganhos adicionais por meio do uso de algoritmos e técnicas mais complexas. Por exemplo, a combinação de ambos os modelos em um *ensemble* pode ser uma alternativa viável para alcançar acurácias ainda melhores. Nesse caso, seria necessário um estudo mais aprofundado para avaliar a sinergia e a complementaridade dos dois modelos.

Em resumo, a escolha do modelo final depende das ne-

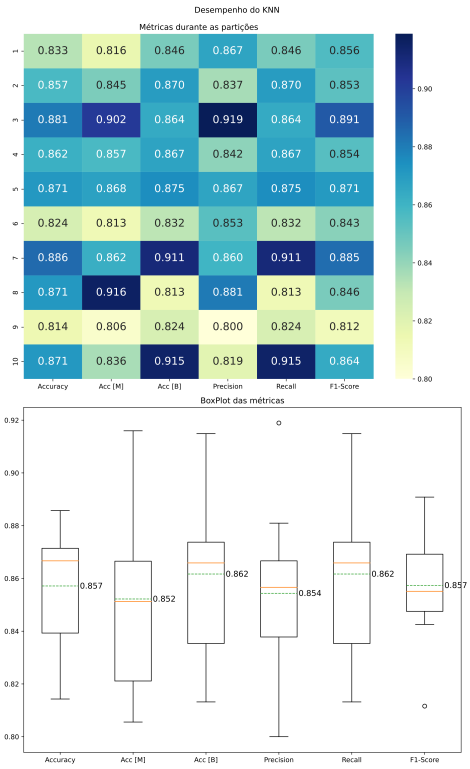

Figure 5: Métricas do SVM

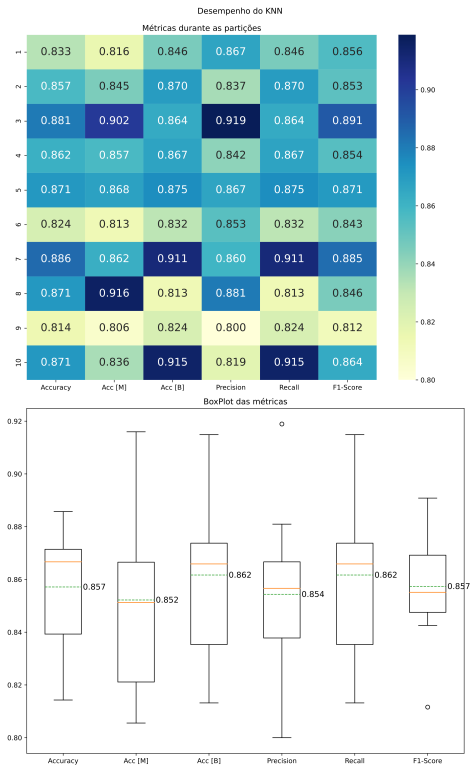

Figure 6: Métricas do Naive Bayes Gaussiano

cessidades específicas da aplicação em termos de acurácia, complexidade e eficiência computacional. A *regressão logística* e o *SVM* são boas opções, e a combinação de diferentes modelos pode ser explorada para obter um desempenho ainda mais aprimorado.

Participação de cada membro

Cecília Hélen Nunes Câmara - Análise exploratória inicial dos dados, treinamento dos modelos de KNN e Random Forest, e escrita do artigo.

Diego Rabelo de Sá - Estruturação do *pipeline* de treinamento, treinamento dos modelos de Regressão Logística e SVM, e escrita do artigo.

Louis Ian Silva dos Santos - Treinamento do modelo Naive Bayes Gaussiano e escrita do artigo.

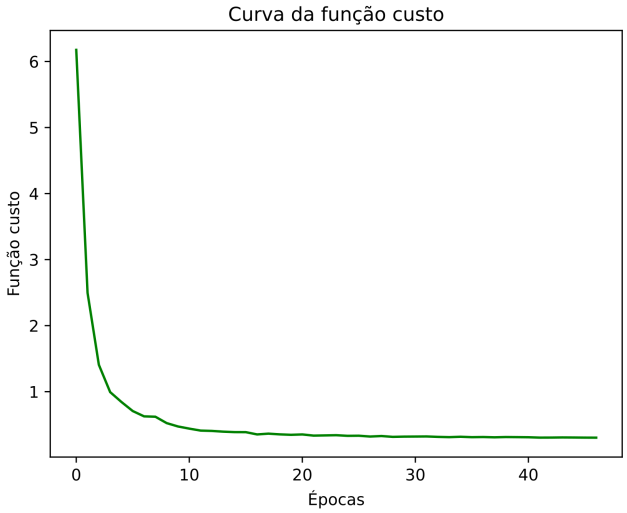

Figure 7: Curva de Aprendizado da Regressão Logística

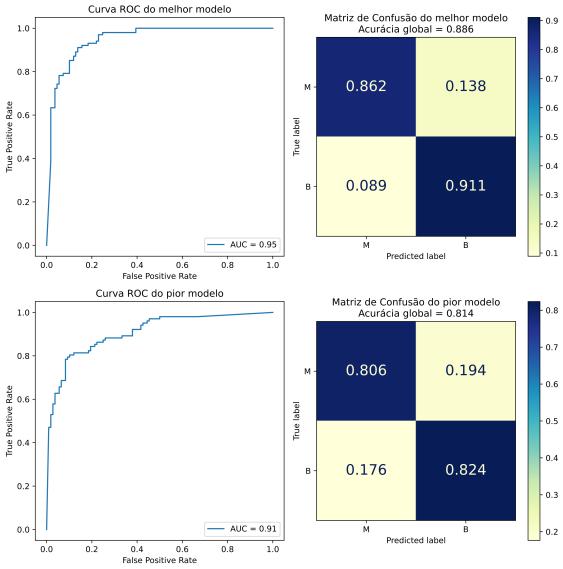

Figure 8: Curva do KNN

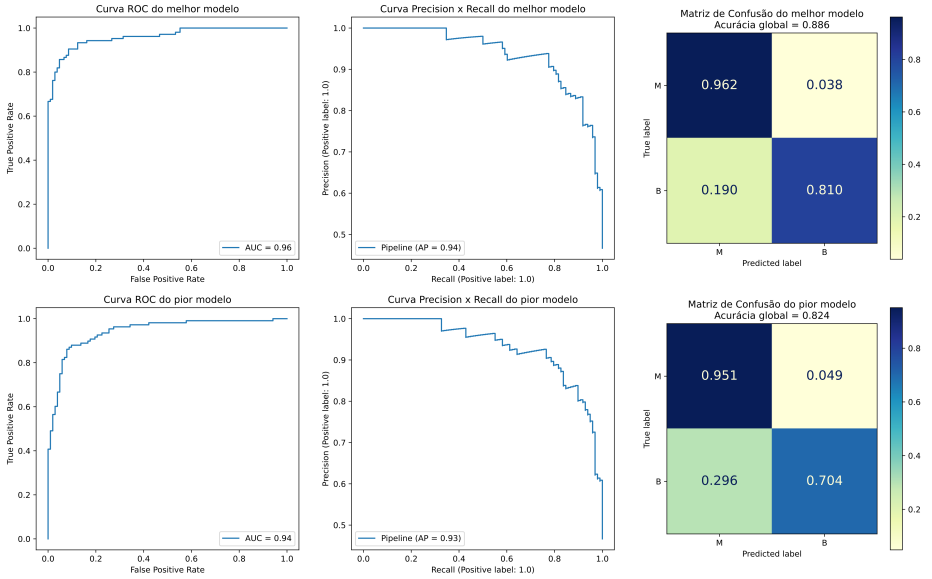

Figure 9: Curvas da Regressão Logística

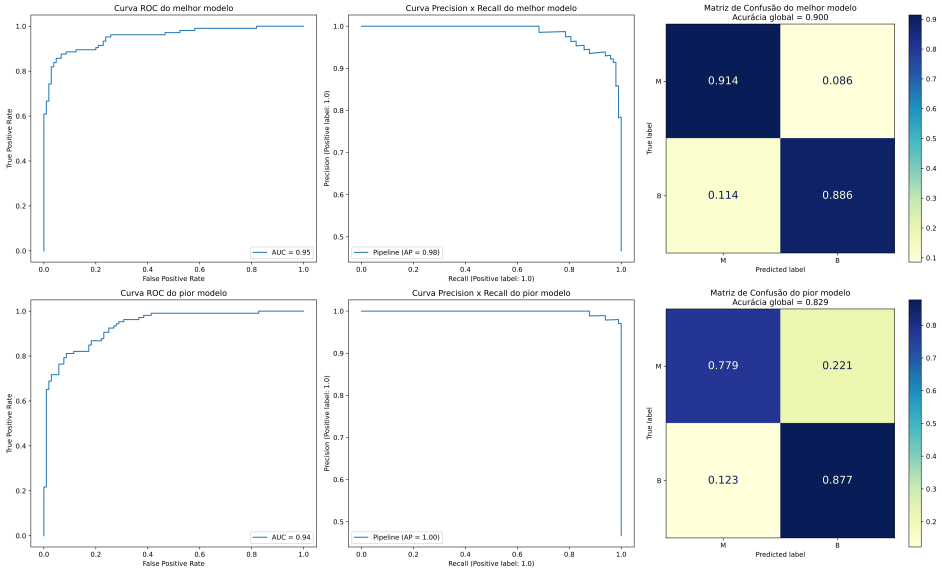

Figure 10: Curvas do Random Forest

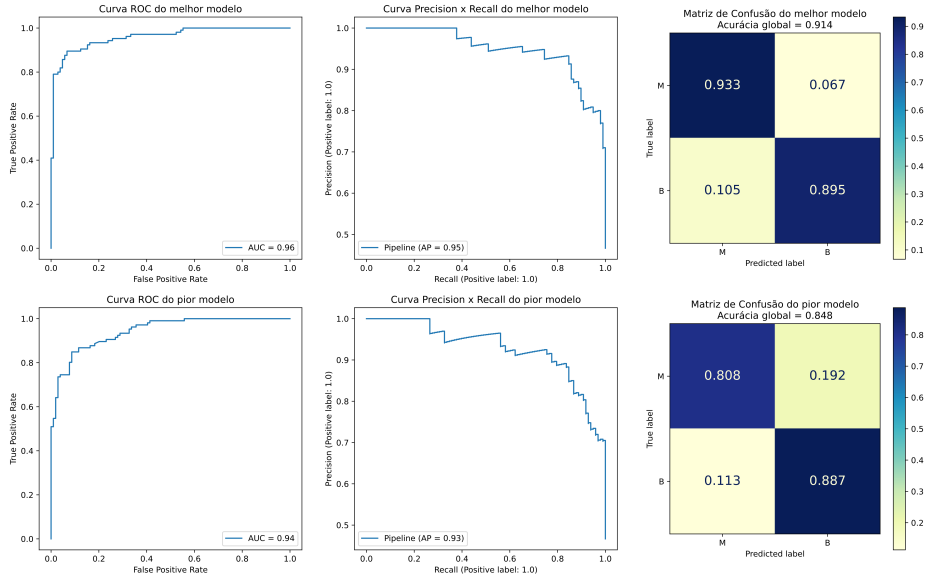

Figure 11: Curvas do SVM

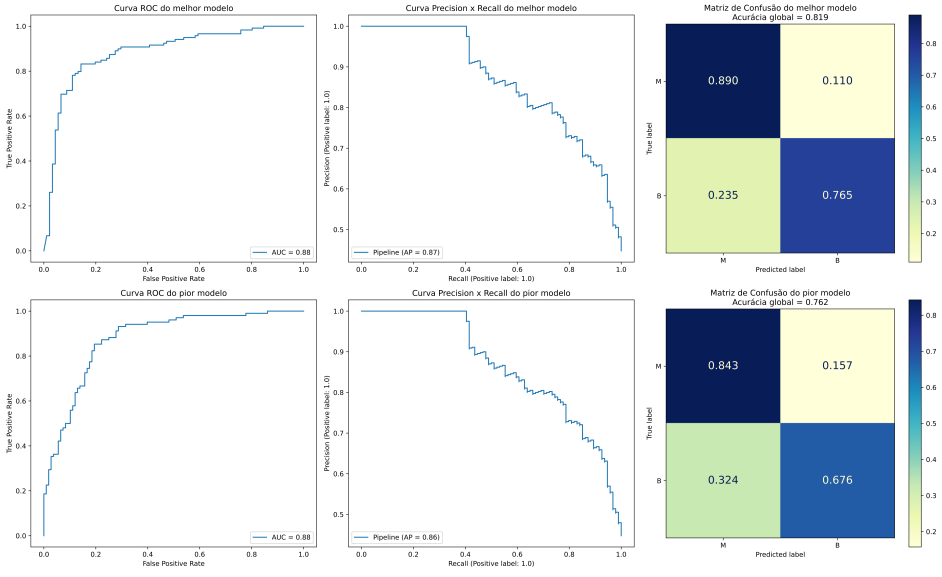

Figure 12: Curvas do Naive Bayes Gaussiano