1 Proposta: Assignment 3 - Fall Detection

O objetivo desta tarefa é desenvolver um classificador capaz de identificar quedas e não-quedas a partir do conjunto de dados disponível neste link. Para mais detalhes sobre um estudo utilizando este conjunto de dados, consulte este artigo.

As aquisições são baseadas no dispositivo IMU Gy-80. Este IMU compreende um acelerômetro triaxial (x,y,z) (modelo ADXL345), um giroscópio triaxial (x,y,z) (modelo L3G4200D) e um magnetômetro triaxial (x,y,z) (modelo HMC5883L), totalizando nove sinais por aquisição. Vinte e dois voluntários participaram, repetindo cada atividade três vezes, totalizando 36 aquisições por ciclo de teste. Assim, torna-se disponível um total de 792 aquisições. Metade delas está relacionada à simulação de quedas (semelhante a este), e a outra metade simula atividades da vida diária. Cada sinal de queda e de não-queda inicia com uma posição estática (braços em repouso), seguida por alguns passos antes da simulação do evento. O conjunto de dados está organizado da seguinte forma:

- Janela de tempo fixa com 5s e $F_s = 100$Hz ($T_s = 0{,}01$s)
- Formato do arquivo: M.S.R.VO.mat (arquivos .mat do MATLAB podem ser abertos em Python com o módulo *scipy.io*, especificamente utilizando a função *loadmat* – exemplo aqui).
  - M: classe do sinal = 1 para não-quedas e 2 para quedas.
  - S: subclasse do sinal (não estamos utilizando esta subclasse nesta tarefa)
    - Para não-quedas, os sinais são: andando; Sentando e levantando de uma cadeira; Batendo palmas; Abrindo e fechando uma porta; Movendo um objeto; Andando + amarrando os sapatos.
    - Para quedas, os sinais são: queda para frente, queda para trás, queda lateral (para o lado do dispositivo), queda lateral (para o lado oposto ao do dispositivo), queda após girar no sentido horário e queda após girar a cintura no sentido anti-horário.
  - R: repetição do movimento, de 1 a 3
  - VO: ID do voluntário, de 01 a 22
- Sensores: As 11 colunas são: tempo, acelerômetro (X, Y e Z), giroscópio (X, Y e Z), magnetômetro (X, Y e Z), e uma coluna zero (use a variável *newData* ao abrir o arquivo).

Comece dividindo o conjunto de dados em treino e teste. Para isso, utilize os voluntários 01–18 para treino e 19–22 para teste. Para simplificar, divida cada conjunto (treinamento e teste) em pastas separadas. Em cada pasta, agrupe os exemplos da classe de queda em uma pasta e os exemplos da classe de não-queda em outra. Dentro de cada arquivo para cada classe, identifique a melhor forma de extrair características dos sinais do acelerômetro (X, Y e Z), giroscópio (X, Y e Z) e magnetômetro (X, Y e Z) e crie um banco de dados de treinamento e teste contendo apenas as características de cada exemplo de queda e de não-queda (ADL).

Características sugeridas: valor máximo, média, desvio padrão, taxa de cruzamento por zero, etc. Por exemplo, se você usar apenas a média de cada sinal, cada exemplo (cada um dos 792 arquivos), contendo 9 sinais com uma janela completa de aquisição, resultará em um vetor com dimensão 9 :

`[mean_acc_X, mean_acc_Y, mean_acc_Z, mean_gyro_X, mean_gyro_Y, mean_gyro_Z, mean_mag_X, mean_mag_Y, mean_mag_Z]`

Mais características irão aumentar a dimensionalidade deste vetor. Uma visualização dos sinais (*plot*) pode ajudar a identificar quais características são mais discriminantes entre quedas e não-quedas.

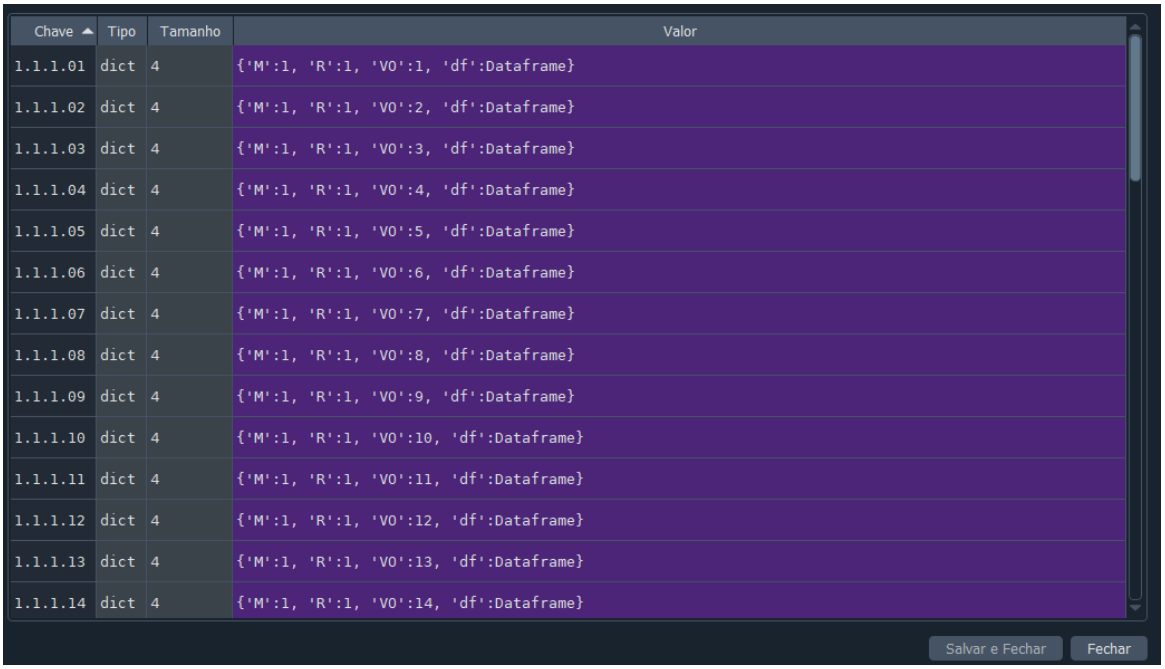
Aplique os classificadores vistos em aula até agora, com busca em grade (*grid search*) e validação cruzada para seleção de hiperparâmetros e análise de curvas para *overfitting*/*underfitting*, verificando os resultados de classificação nos conjuntos de treino e teste. Inclua os resultados e destaques *aqui*.

2 Resolução

Neste *assignment*, as *features* foram extraídas a partir dos sinais capturados pelos sensores inerciais: acelerômetro, giroscópio e magnetômetro. Para cada eixo (X, Y e Z) de cada sensor, foram computadas as seguintes estatísticas temporais:

- Média (Mean): valor médio do sinal ao longo do tempo;
- Desvio padrão (Standard Deviation): medida da dispersão em torno da média;
- Variância (Variance): quadrado do desvio padrão, representa a energia da variabilidade;
- Valor mínimo (Minimum): menor valor observado no sinal;
- Valor máximo (Maximum): maior valor observado no sinal;
- Amplitude (Range): diferença entre os valores máximo e mínimo;
- Valor RMS (Root Mean Square): raiz da média dos quadrados dos valores, útil para avaliar a energia do sinal.

Os dados de treino/teste foram separados conforme o código do arquivo `ler_arquivos.py`.



Chave	Tipo	Tamanho	Valor
1.1.1.01	dict	4	{'M':1, 'R':1, 'VO':1, 'df':Dataframe}
1.1.1.02	dict	4	{'M':1, 'R':1, 'VO':2, 'df':Dataframe}
1.1.1.03	dict	4	{'M':1, 'R':1, 'VO':3, 'df':Dataframe}
1.1.1.04	dict	4	{'M':1, 'R':1, 'VO':4, 'df':Dataframe}
1.1.1.05	dict	4	{'M':1, 'R':1, 'VO':5, 'df':Dataframe}
1.1.1.06	dict	4	{'M':1, 'R':1, 'VO':6, 'df':Dataframe}
1.1.1.07	dict	4	{'M':1, 'R':1, 'VO':7, 'df':Dataframe}
1.1.1.08	dict	4	{'M':1, 'R':1, 'VO':8, 'df':Dataframe}
1.1.1.09	dict	4	{'M':1, 'R':1, 'VO':9, 'df':Dataframe}
1.1.1.10	dict	4	{'M':1, 'R':1, 'VO':10, 'df':Dataframe}
1.1.1.11	dict	4	{'M':1, 'R':1, 'VO':11, 'df':Dataframe}
1.1.1.12	dict	4	{'M':1, 'R':1, 'VO':12, 'df':Dataframe}
1.1.1.13	dict	4	{'M':1, 'R':1, 'VO':13, 'df':Dataframe}
1.1.1.14	dict	4	{'M':1, 'R':1, 'VO':14, 'df':Dataframe}

Salvar e Fechar Fechar

Figura 1: Variável `dados`, do tipo dicionário.

Após a leitura bruta dos dados e armazenadas no dicionário, separou-se em `dados_train` e `dados_teste`: dessa forma os voluntários 01-18 vão ser utilizados para treino e os 19-22 utilizados para teste final.

```
# Divide o dicionário principal em treino e teste
dados_train = {k: v for k, v in dados.items() if v['VO'] <= 18}
dados_test = {k: v for k, v in dados.items() if v['VO'] >= 19}
```

Em seguida criou-se a função `extrair_features` para calcular as *features* apresentadas anteriormente.

```
# Função para extrair estatísticas de um exemplo (ignorando a coluna de tempo)
def extrair_features(exemplo):
    df = exemplo['df']

    # Inicializa dicionário com rótulo (queda ou não) e identificadores
    features = {
        'Fall': 'yes' if exemplo['M'] == 2 else 'no',  # Define rótulo de queda: M=2 ind
    }

    # Para cada sensor (aceleração, giroscópio, magnetômetro nos eixos X, Y, Z)
    for col in colunas[1:]:  # Ignora a coluna 'tempo'
        x = df[col].astype(np.float64)  # Garante precisão numérica
        # Extrai estatísticas básicas da série temporal
        features[f'{col}_mean'] = x.mean()                # Média
        features[f'{col}_std'] = x.std()                  # Desvio padrão
        features[f'{col}_var'] = x.var()                  # Variância
        features[f'{col}_min'] = x.min()                  # Mínimo
        features[f'{col}_max'] = x.max()                  # Máximo
        features[f'{col}_range'] = x.max() - x.min()      # Amplitude
        features[f'{col}_rms'] = np.sqrt(np.mean(x**2))   # RMS (root mean square)

    return features
# Extrai features para treino
features_train = [extrair_features(ex) for ex in dados_train.values()]
df_train = pd.DataFrame(features_train)

# Extrai features para teste
features_test = [extrair_features(ex) for ex in dados_test.values()]
df_test = pd.DataFrame(features_test)
```

Após a extração das *features*, o **dataframe** `df_train` resultou em 648 exemplos, enquanto o `df_test` ficou com 144 exemplos. Isso corresponde a uma proporção de aproximadamente 82% para treino e 18% para teste.

Scoring escolhido para o `Grid_SearchCV`

Neste problema de detecção de quedas em voluntários, optou-se por dar maior importância à capacidade do modelo em identificar corretamente os casos de queda. Ou seja, o objetivo principal é **minimizar os falsos negativos** (quedas não detectadas), mesmo que isso possa acarretar um aumento nos falsos positivos.

Para isso, foi adotado como métrica de otimização o escore F_β, que representa a **média harmônica ponderada entre *precision* e *recall***. Esse *score* assume valor ideal em 1 (classificação perfeita) e valor mínimo em 0.

O parâmetro β permite ajustar a importância relativa entre *recall* e *precision*:

- $\beta > 1$: dá mais peso ao *recall* (detecta mais quedas, tolerando falsos positivos);

- $\beta < 1$: favorece o *precision* (evita alarmes falsos, mas pode perder quedas reais);
- $\beta = 1$: corresponde ao F_1-score, onde precision e recall têm igual importância;
- Assintoticamente: $\beta \to \infty$ considera apenas o recall; $\beta \to 0$ considera apenas o precision.

A fórmula do escore F_β é:

$$F_\beta = \frac{(1+\beta^2)\,\mathrm{TP}}{(1+\beta^2)\,\mathrm{TP} + \beta^2\,\mathrm{FN} + \mathrm{FP}} \tag{1}$$

Onde:

- tp: número de verdadeiros positivos (quedas corretamente detectadas);
- fp: número de falsos positivos (não-quedas classificadas como quedas);
- fn: número de falsos negativos (quedas não detectadas).

No presente trabalho, escolheu-se[1] $\beta = 2$, priorizando o *recall* com o dobro da importância em relação à *precision*. Essa escolha está alinhada com o objetivo de reduzir o risco de quedas não identificadas.
Embora as classes estejam aproximadamente balanceadas no conjunto de dados, optou-se por utilizar pesos diferentes para as classes nos classificadores com suporte a ponderação, a fim de refletir a importância prática da classe de queda (1).

2.1 Adaline (com SGDClassifier)

2.1.1 Desempenho variando o parâmetro de regularização alpha

➢ Os resultados obtidos pelo arquivo: `ada_validation_curve.py`.

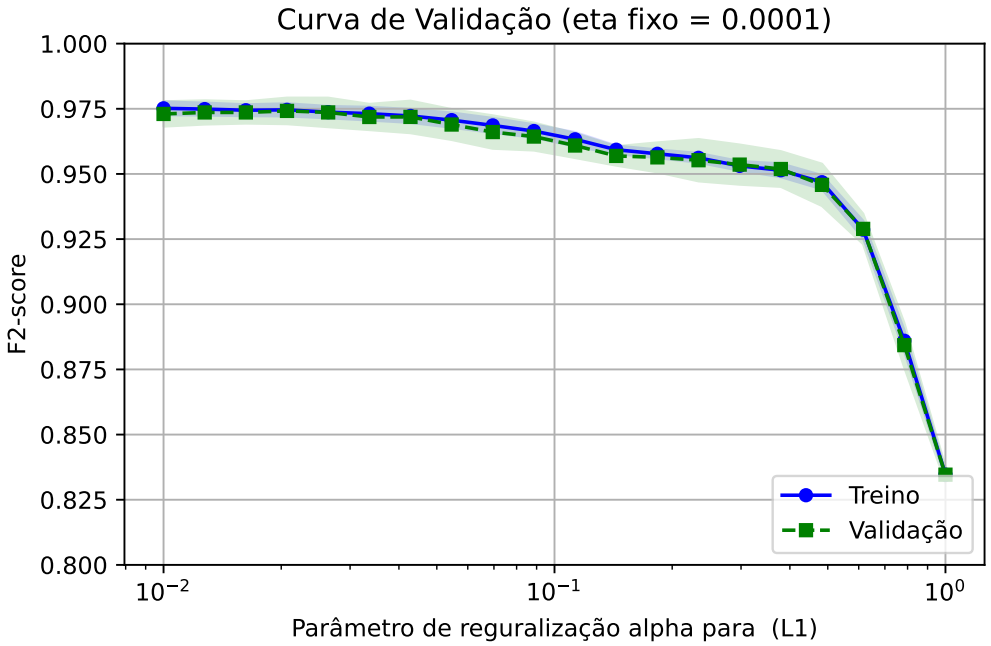


Figura 2: Adaline com regularização L1: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de regularização α.

[1]Por esse motivo vamos denotar F_2-score para um score F_β com $\beta = 2$

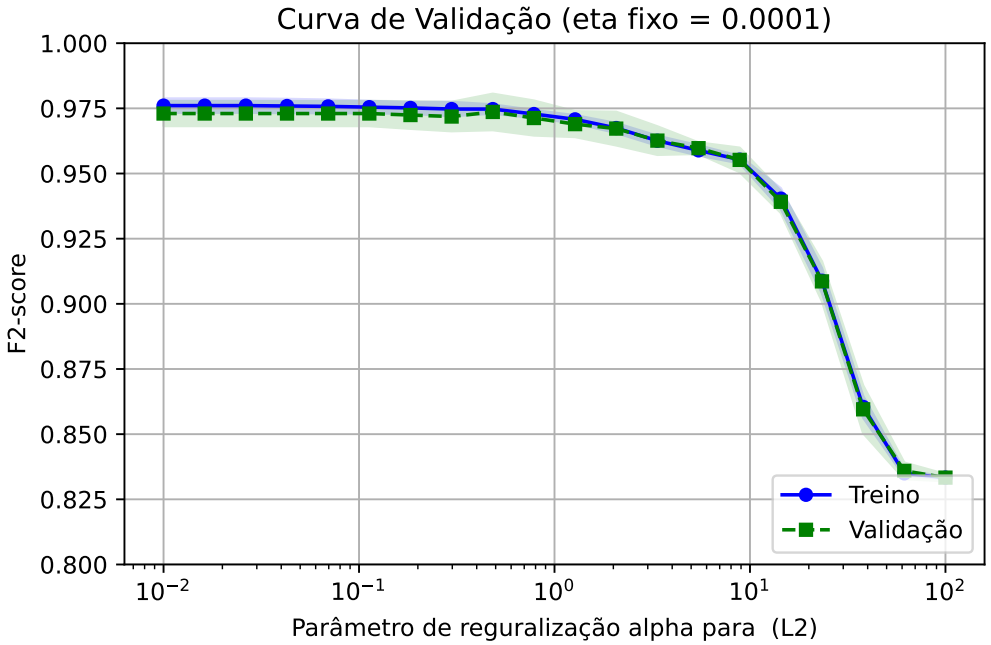


Figura 3: Adaline com regularização L2: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de regularização α.

2.1.2 Resultados no conjunto de treino e teste

Os resultados do classificador Adaline, foram obtidos por meio do arquivo: `adaline_SGD.py`.

- **Melhor F_2-score médio no** `GridSearchCV`: 0,975
- **Melhores hiperparâmetros encontrados pelo** `GridSearchCV`:
  - `sgdclassifier_alpha`: 0,001
  - `sgdclassifier_penalty`: L1

Avaliação final no conjunto de teste (voluntários com `VO` de 19 a 22):

- **F_2-score**: 0,973
- **Precision** : 0,986
- **Recall** : 0,972
- **F_1-score**: 0,979
- **Coeficiente de Correlação de Matthews (MCC)**: 0,958

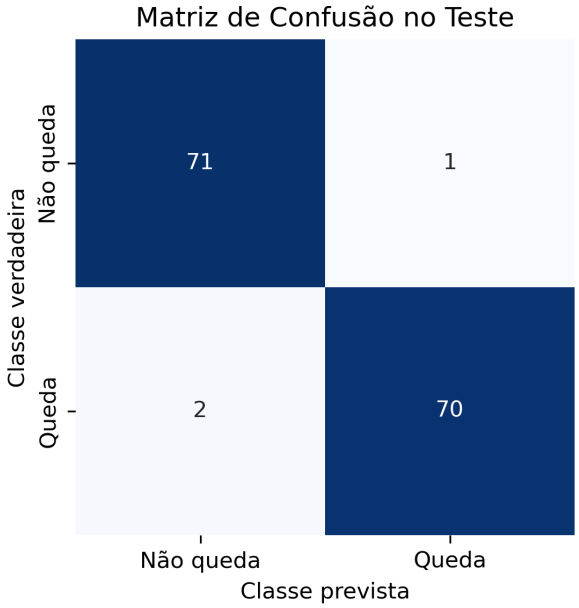


Figura 4: Adaline: Matriz de confusão do conjunto de teste final.

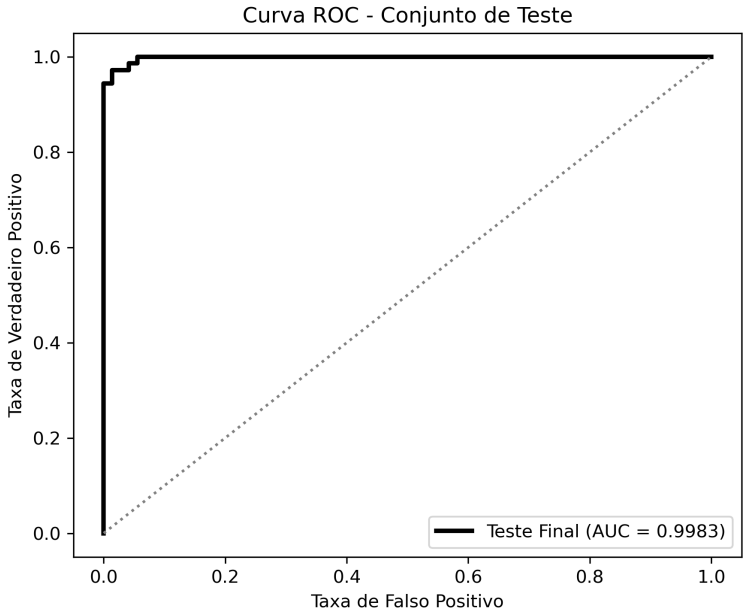


Figura 5: Adaline: Curva ROC do conjunto de teste final.

2.2 Random Forest

2.2.1 Desempenho variando os hiperparâmetros `max_depth` e `n_estimators`

➢ Resultados obtidos pelo arquivo random_forest_validation.py

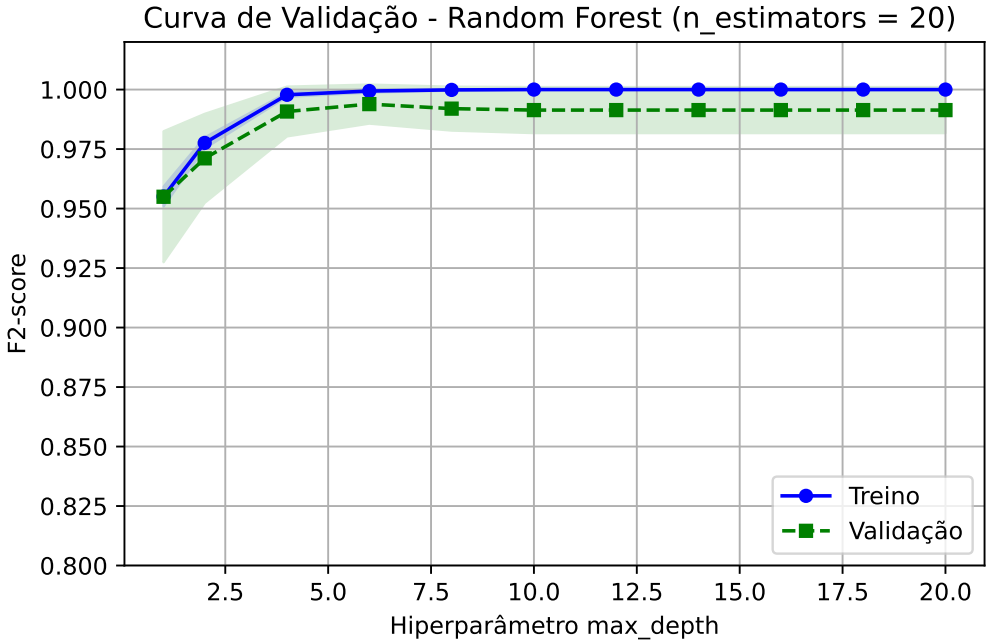


Figura 6: Random Forest: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de profundidade máxima da árvore (`max_depth`) para um valor fixo de `n_estimators`= 20 .

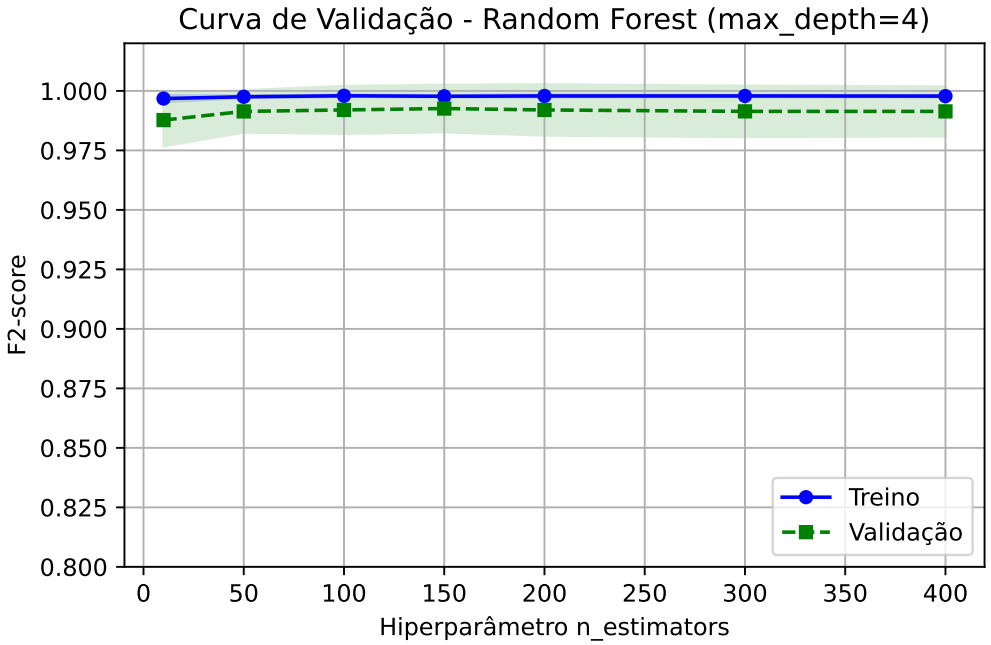


Figura 7: Random Forest: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de número máximo de árvores (`n_estimators`) para um valor fixo de `max_depth`= 4 .

2.2.2 Resultados no conjunto de treino e teste

Os resultados obtidos do classificador Random Forest, foram obtidos por meio do arquivo: `random_forest.py`.

- **Melhor F_2-score médio no `GridSearchCV`**: 0,9907993868950425
- **Melhores hiperparâmetros encontrados pelo `GridSearchCV`**:
  - `randomforestclassifier__n_estimators`: 100
  - `randomforestclassifier__max_depth`: 4
  - `randomforestclassifier__min_samples_leaf`: 2
  - `randomforestclassifier__ccp_alpha`: 0.001.

Avaliação no conjunto de teste final (voluntários com `VO` de 19 a 22):

- **Acurácia**: 0,972
- **F_2-score**: 0,955
- **Precision** : 1,000
- **Recall** : 0,944
- **F_1-score**: 0,971
- **Coeficiente de Correlação de Matthews (MCC)**: 0,946
- **F_2-score (teste)**: 0,948

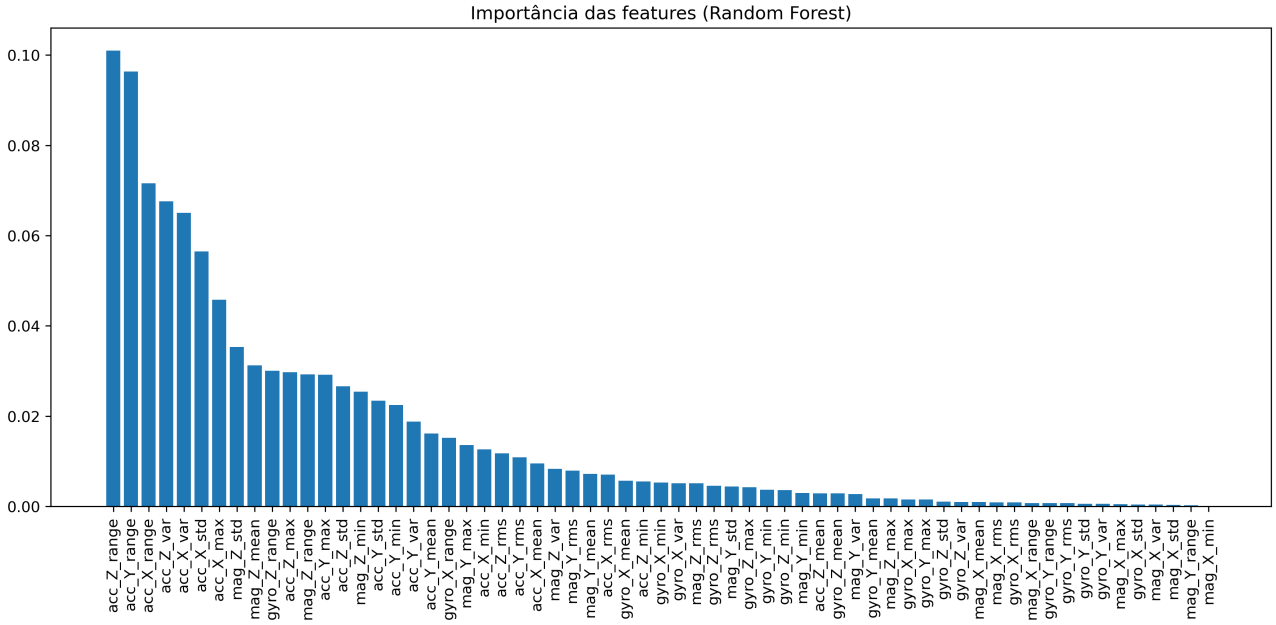


Figura 8: Random Forest: Importância das *features*

Tabela 1: Importância das *features* no modelo (ordem decrescente)

	Feature		*Feature*		*Feature*
1	acc_Z_range	22	acc_X_min	43	mag_Y_var
2	acc_Y_range	23	acc_Z_rms	44	gyro_Y_mean
3	acc_X_range	24	acc_Y_rms	45	mag_Z_max
4	acc_Z_var	25	acc_X_mean	46	gyro_X_max
5	acc_X_var	26	mag_Z_var	47	gyro_Y_max
6	acc_X_std	27	mag_Y_rms	48	gyro_Z_std
7	acc_X_max	28	mag_Y_mean	49	gyro_Z_var
8	mag_Z_std	29	acc_X_rms	50	mag_X_mean
9	mag_Z_mean	30	gyro_X_mean	51	mag_X_rms
10	gyro_Z_range	31	acc_Z_min	52	gyro_X_rms
11	acc_Z_max	32	gyro_X_min	53	mag_X_range
12	mag_Z_range	33	gyro_X_var	54	gyro_Y_range
13	acc_Y_max	34	mag_Z_rms	55	gyro_Y_rms
14	acc_Z_std	35	gyro_Z_rms	56	gyro_Y_std
15	mag_Z_min	36	mag_Y_std	57	gyro_Y_var
16	acc_Y_std	37	gyro_Z_max	58	mag_X_max
17	acc_Y_min	38	gyro_Y_min	59	gyro_X_std
18	acc_Y_var	39	gyro_Z_min	60	mag_X_var
19	acc_Y_mean	40	mag_Y_min	61	mag_X_std
20	gyro_X_range	41	acc_Z_mean	62	mag_Y_range
21	mag_Y_max	42	gyro_Z_mean	63	mag_X_min

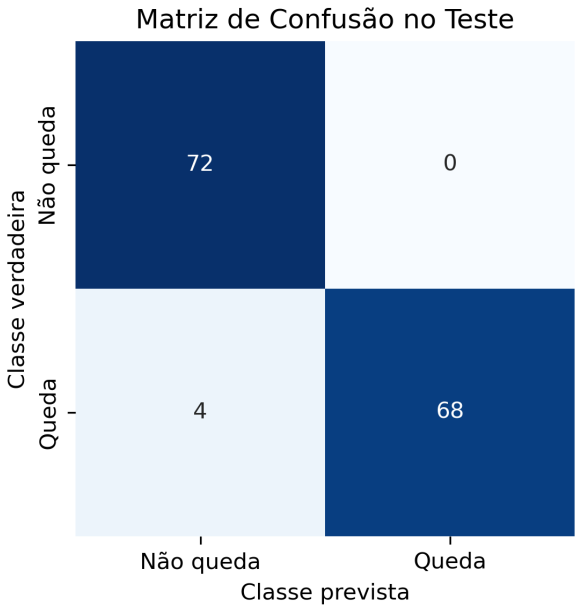


Figura 9: Random Forest: Matriz de confusão do conjunto de teste final.

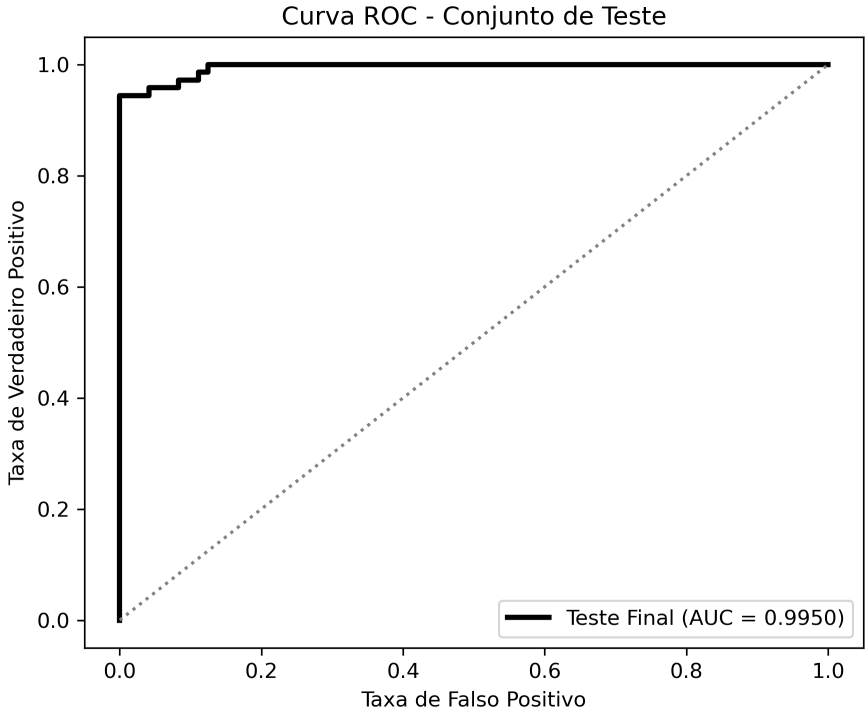


Figura 10: Random Forest: Curva ROC do conjunto de teste final.

2.3 K-nearest neighbors (Knn)

O classificador *K-Nearest Neighbors* (KNN) é particularmente suscetível ao *overfitting* em cenários de alta dimensionalidade, fenômeno conhecido como *maldição da dimensionalidade*. À medida que o número de atributos cresce, a noção de distância — fundamental para o funcionamento do KNN — torna-se menos discriminativa, dificultando a distinção entre vizinhos próximos e distantes.

Além disso, assim como ocorre com algoritmos baseados em árvores de decisão, o KNN não possui um mecanismo explícito de regularização para controlar a complexidade do modelo. Dessa forma, torna-se essencial empregar técnicas de *feature selection* ou de redução de dimensionalidade, como a *Análise de Componentes Principais* (PCA), para mitigar o sobreajuste e melhorar a generalização do modelo.

Diante desse cenário, utilizou-se o objeto `PCA` da biblioteca *scikit-learn*, previamente ajustado aos dados de treino. Os autovalores, que representam a variância explicada por cada componente principal, encontram-se armazenados na variável `pca.explained_variance_`, enquanto os autovetores — responsáveis pelas direções dos eixos principais no novo espaço projetado — estão disponíveis em `pca.components_`. Para fins de consistência na orientação dos vetores, utilizou-se a transposta multiplicada por -1, i.e., `(-1) * pca.components_.T`. A Figura 11 apresenta a proporção da variância total explicada por cada componente principal [2]

[2]O trecho de código que realiza esse gráfico está no arquivo ler_arquivos.py

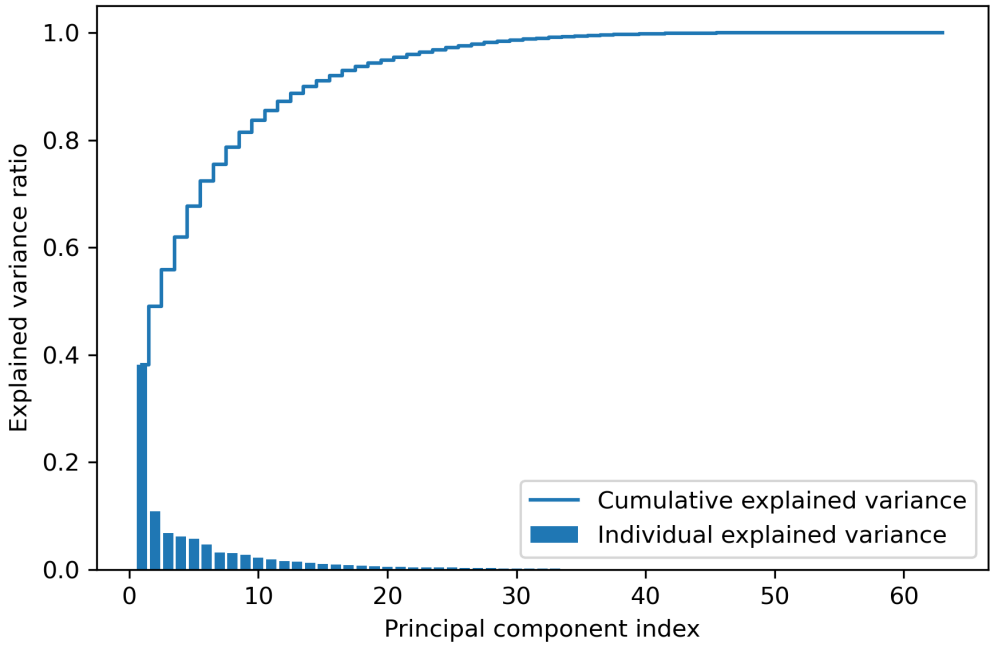


Figura 11: A proporção da variância total capturada pelos componentes principais

2.3.1 Verificando o impacto da variação hyperparâmetro k com PCA(15)

➢ Resultados obtidos pelo arquivo knn_validation.py

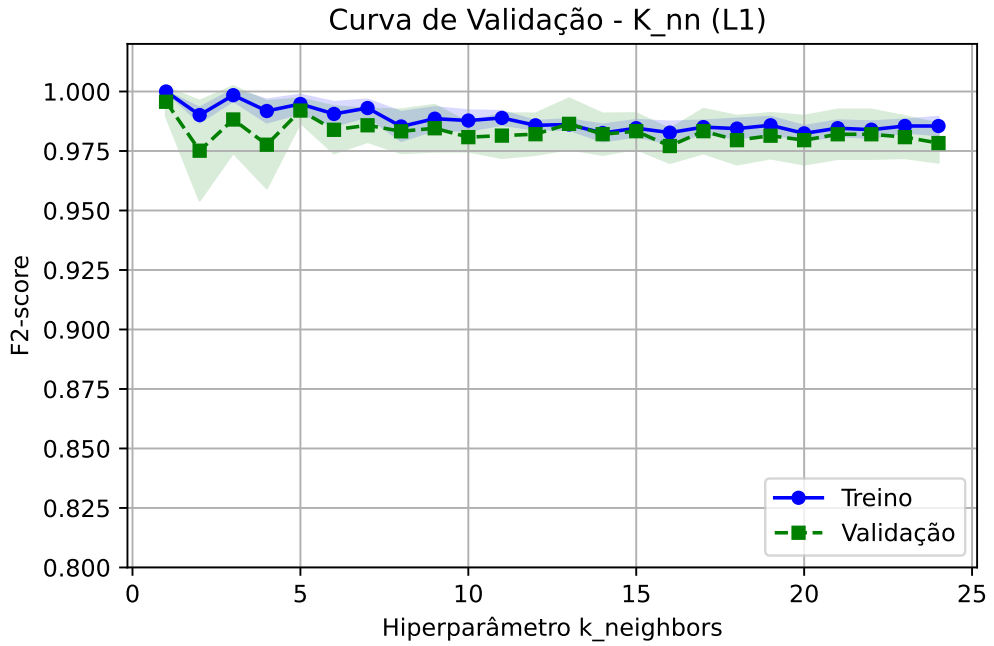


Figura 12: Knn: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de quantidade de vizinhos mais próximos **k** (`n_neighbors`).As distâncias foram calculadas utilizando a métrica de Manhattan (L1) e PCA(15).

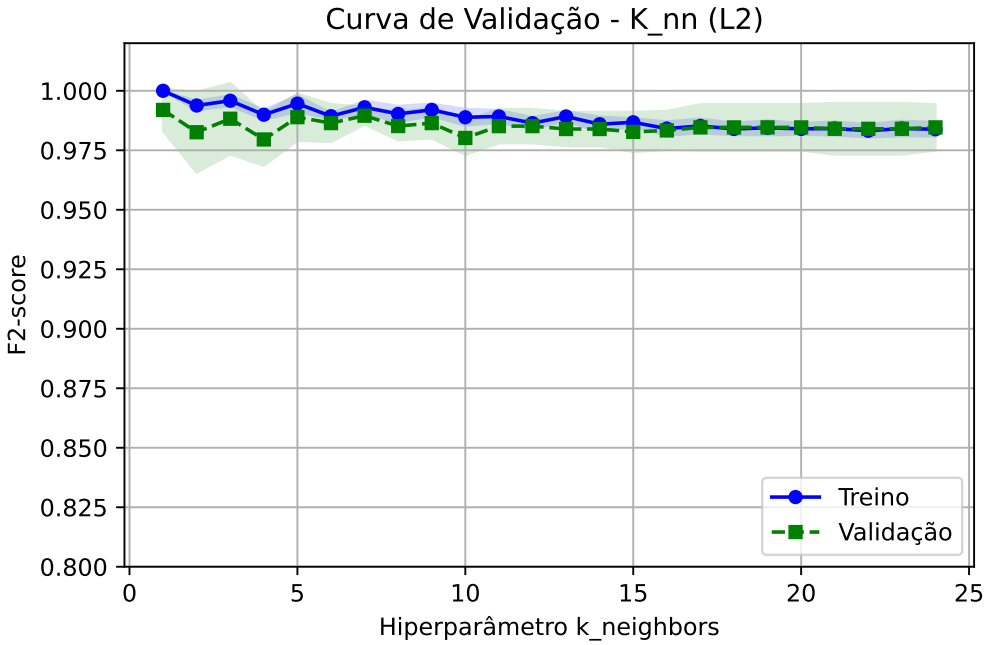


Figura 13: Knn: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro de quantidade de vizinhos mais próximos **k** (`n_neighbors`).As distâncias foram calculadas utilizando a métrica Euclidiana (L2) e PCA(15).

2.3.2 Resultados no conjunto de treino e teste

Os resultados do classificador Knn, foram obtidos por meio do arquivo: `knn.py`.

- **Melhor F_2-score médio no** `GridSearchCV`: 0,9833976746806424
- **Melhores hiperparâmetros encontrados pelo** `GridSearchCV`:
  - `kneighborsclassifier__n_neighbors`: 13
  - `kneighborsclassifier__weights`: uniform
  - `kneighborsclassifier__p`: 2 (L2)

Avaliação no conjunto de teste final (voluntários com `VO` de 19 a 22):

- **Acurácia**: 0,958
- **F_2-score**: 0,932
- **Precision** : 1,000
- **Recall** : 0,917
- **F_1-score**: 0,957
- **Coeficiente de Correlação de Matthews (MCC)**: 0,920
- **F_2-score (teste)**: 0,921

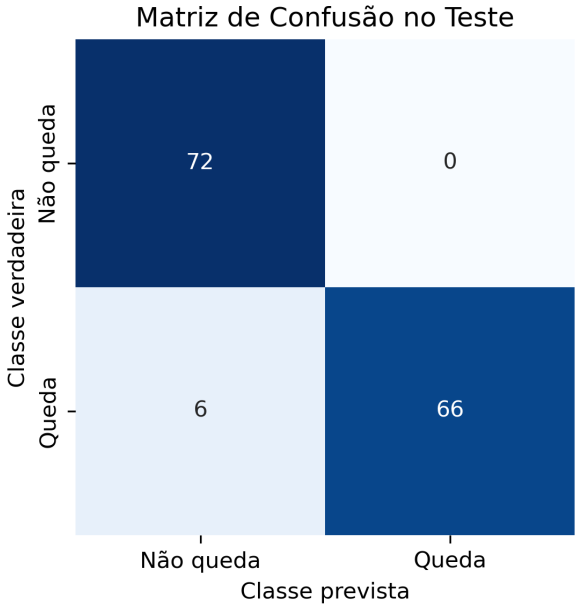


Figura 14: Knn: Matriz de confusão do conjunto de teste final.

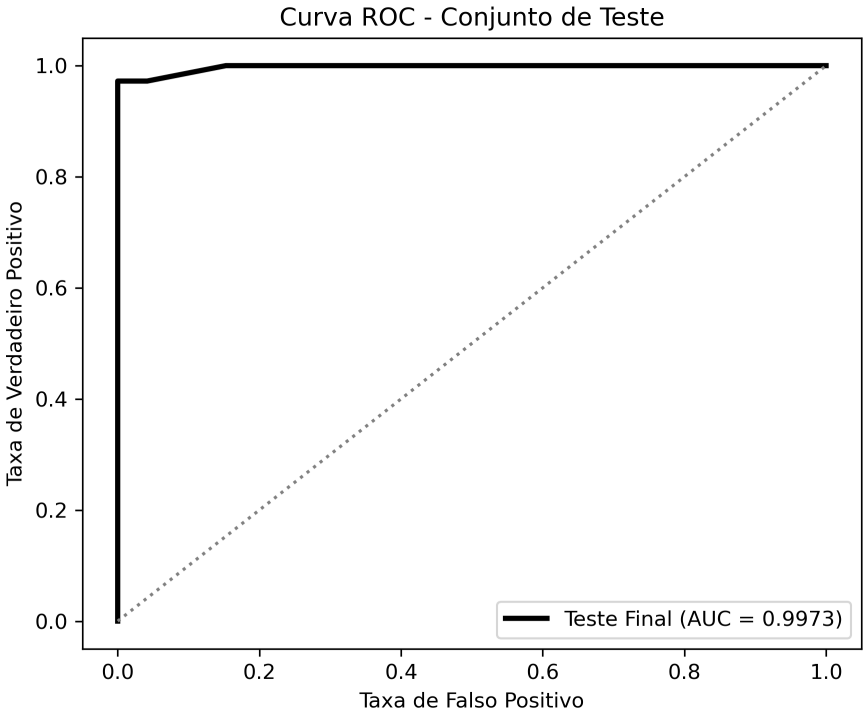


Figura 15: Knn: Curva ROC do conjunto de teste final.

2.4 Logistic Regression

2.4.1 Desempenho variando o hiperparâmetro C

➢ Resultados obtidos pelo arquivo logistic_regression_validation.py

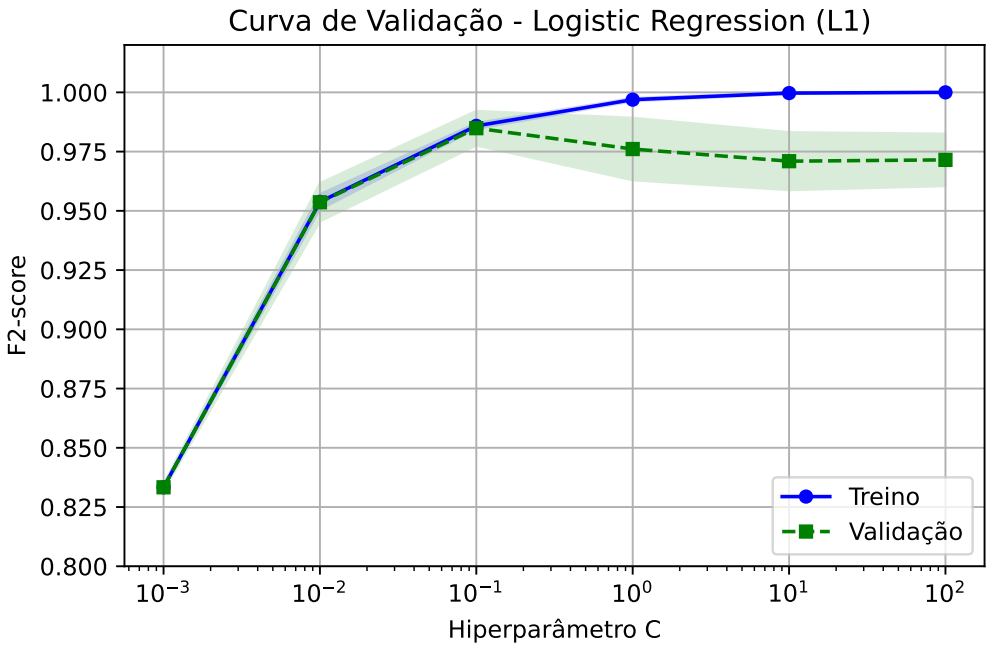


Figura 16: Logistic Regression: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro C para regularização **L1**.

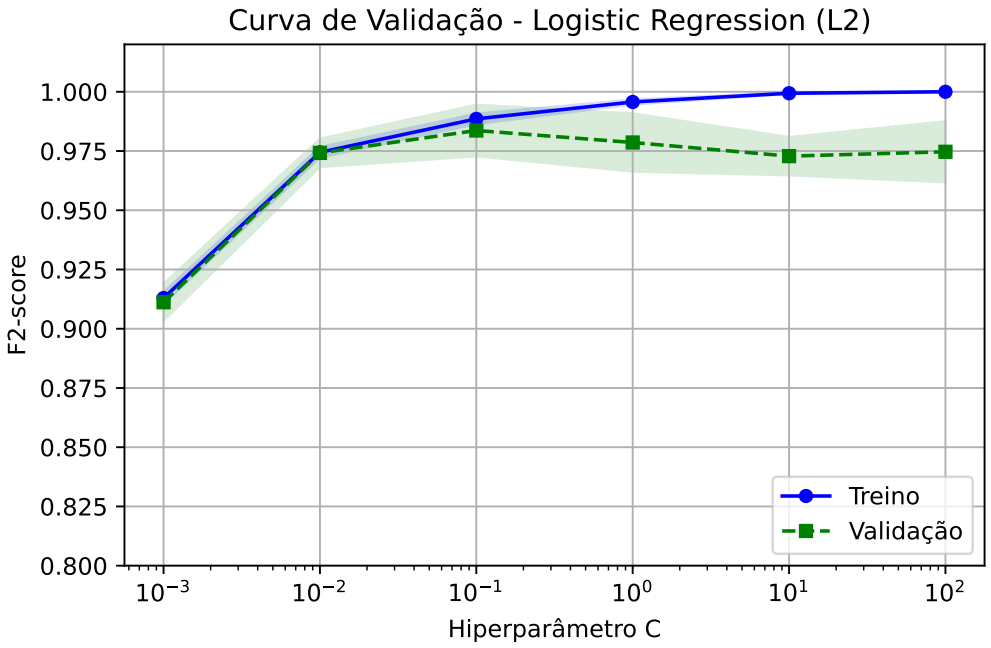


Figura 17: Logistic Regression: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro C para regularização **L2**.

2.4.2 Resultados no conjunto de treino e teste

Os resultados do classificador Logistic Regression, foram obtidos por meio do arquivo: `logistic_regression.py`.

- **Melhor F_2-score médio no `GridSearchCV`**: 0,985931244096909
- **Melhores hiperparâmetros encontrados pelo `GridSearchCV`**:
  - `logisticregression__C`: 0,2
  - `logisticregression__penalty`: l2
  - `logisticregression__solver`: lbfgs

Avaliação no conjunto de teste final (voluntários com `VO` de 19 a 22):

- **Acurácia**: 0,972
- **F_2-score**: 0,964
- **Precision** : 0,986
- **Recall** : 0,958
- **F_1-score**: 0,972
- **Coeficiente de Correlação de Matthews (MCC)**: 0,945
- **F_2-score (teste)**: 0,960

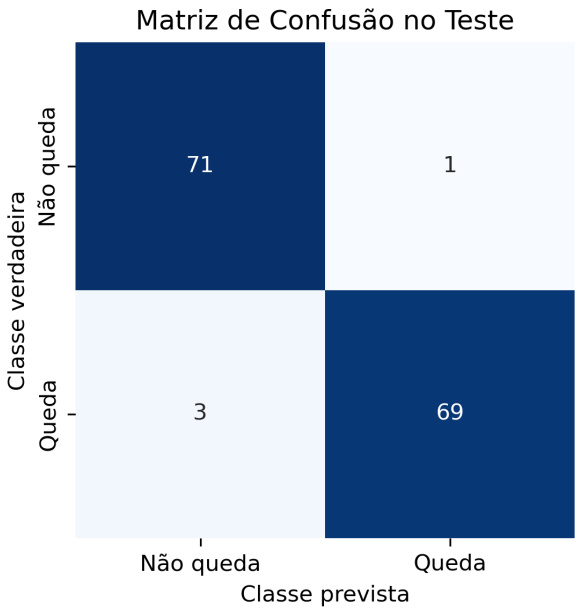


Figura 18: Logistic Regression: Matriz de confusão do conjunto de teste final.

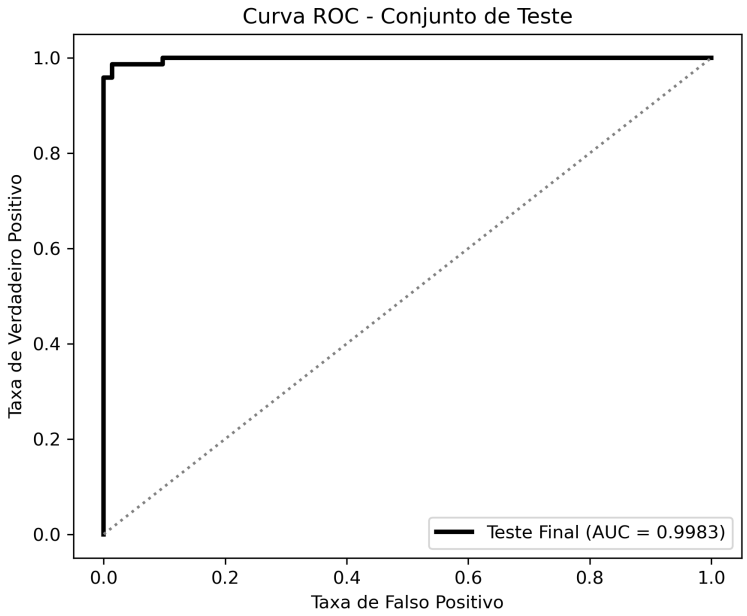


Figura 19: Logistic Regression: Curva ROC do conjunto de teste final.

2.5 SVM (Support Vector Machine)

2.5.1 Desempenho variando os hiperparâmetros `kernel`, `C` e `alpha`

➢ Resultados obtidos pelo arquivo svm_validation_curve.py

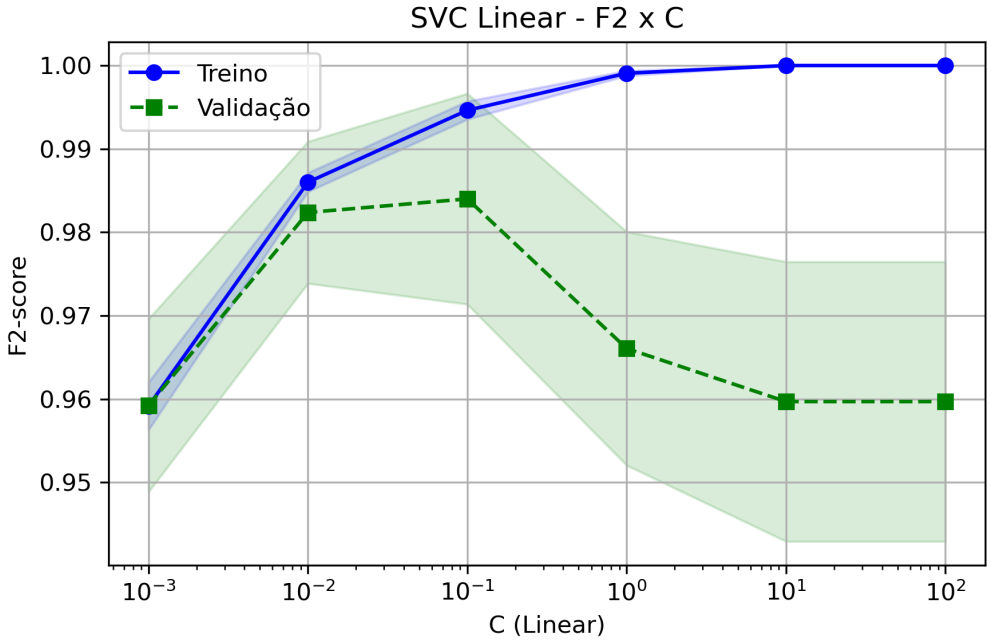


Figura 20: SVM: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro C para `kernel=linear`.

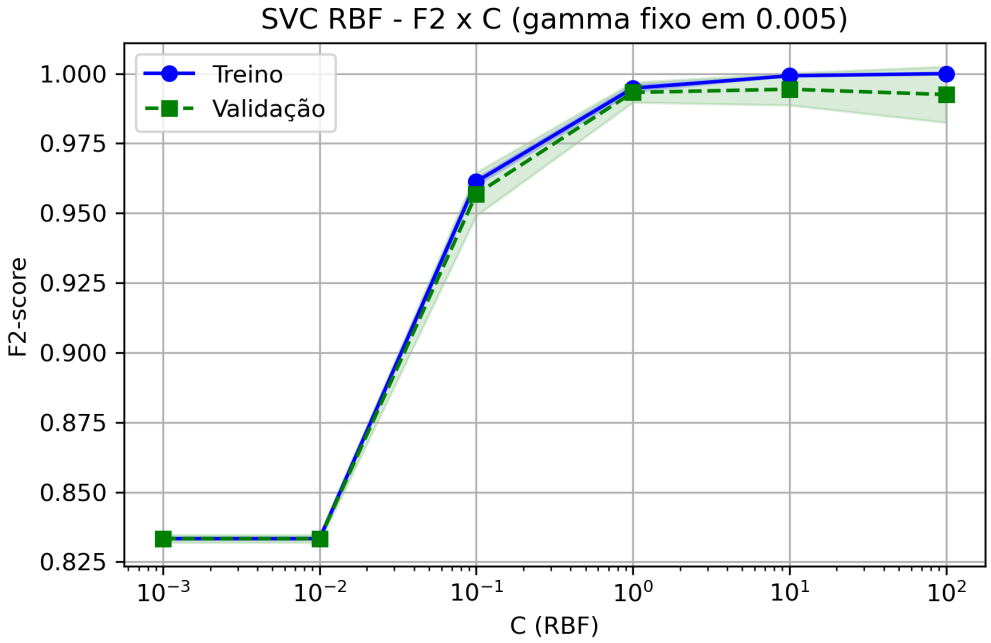


Figura 21: SVM: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro gamma com `kernel=rbf`.

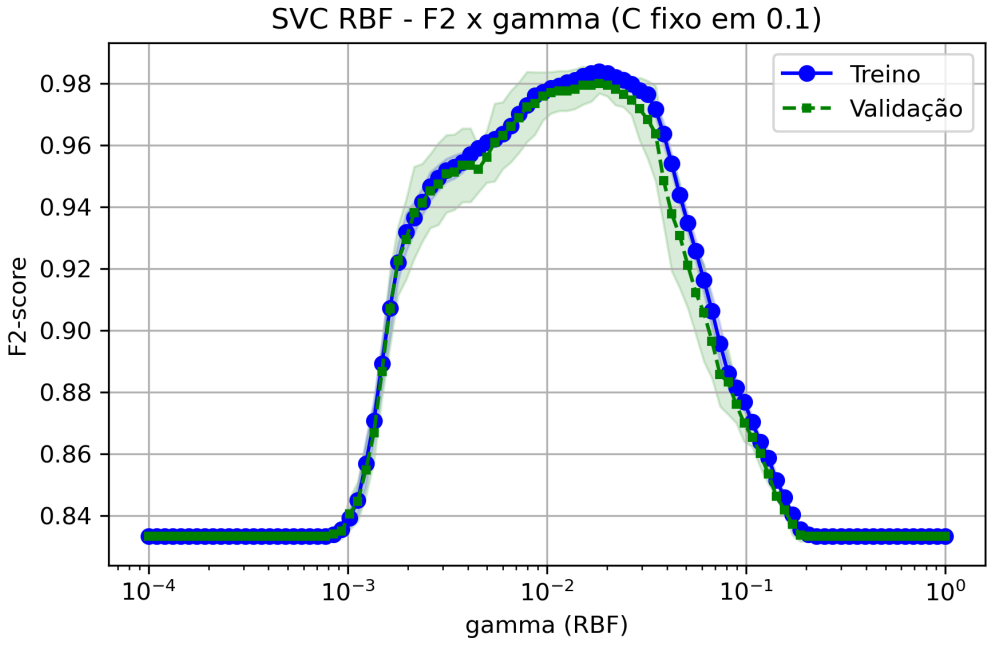


Figura 22: SVM: desempenho do modelo (mean F_2-score) com 5-fold cross-validation para diferentes valores do hiperparâmetro gamma com `kernel=rbf` e `C=0.1`.

2.5.2 Resultados no conjunto de treino e teste

Os resultados do classificador SVM, foram obtidos por meio do arquivo: `svm.py`.

- **Melhor F_2-score médio no `GridSearchCV`**: 0,9831018783358149
- **Melhores hiperparâmetros encontrados pelo `GridSearchCV`**:
  - `svc__C`: 0,9
  - `svc__gamma`: 0,003
  - `svc__kernel`: rbf

Avaliação no conjunto de teste final (voluntários com `VO` de 19 a 22):

- **Acurácia**: 0,979
- **F_2-score**: 0,975
- **Precision** : 0,986
- **Recall** : 0,972
- **F_1-score**: 0,979
- **Coeficiente de Correlação de Matthews (MCC)**: 0,958

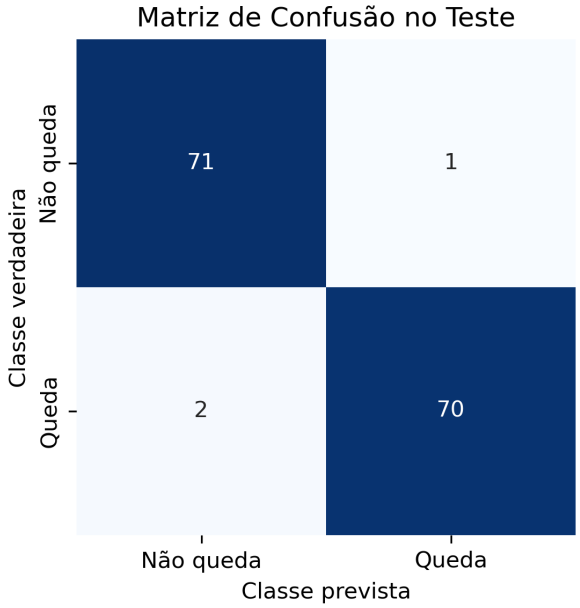


Figura 23: SVM: Matriz de confusão do conjunto de teste final.

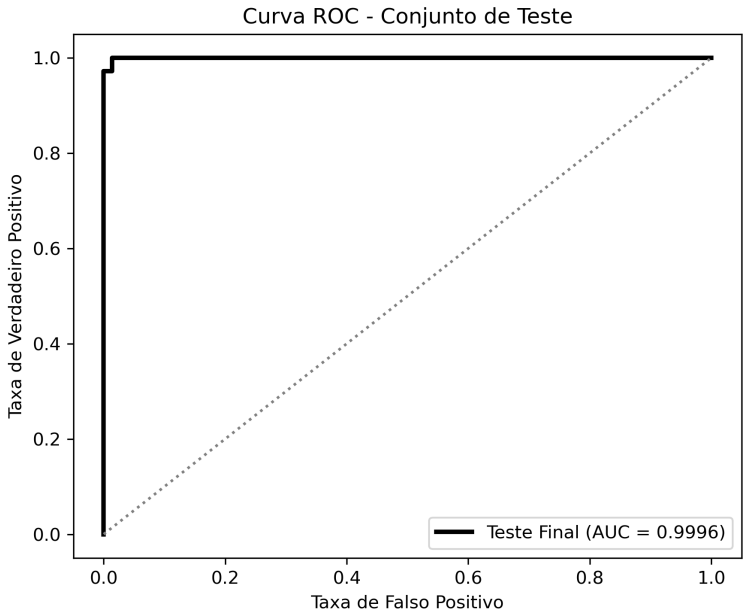


Figura 24: SVM: Curva ROC do conjunto de teste final.

Como o SVM apresentou os melhores resultados entre os modelos avaliados, foram realizadas análises adicionais utilizando os mesmos splits da `cross-validation` do `GridSearchCV` e os melhores hiperparâmetros encontrados. Para cada divisão (*fold*), foram geradas a matriz de confusão e a curva ROC, permitindo avaliar o desempenho do modelo em cada partição.

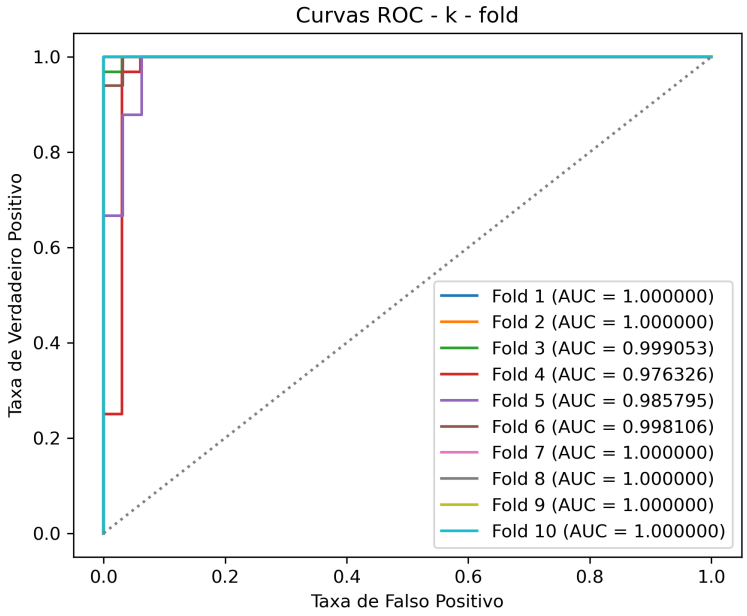


Figura 25: SVM: Curvas ROC obtidas para cada um dos 10 folds da validação cruzada, utilizando os melhores hiperparâmetros encontrados via `GridSearchCV`.

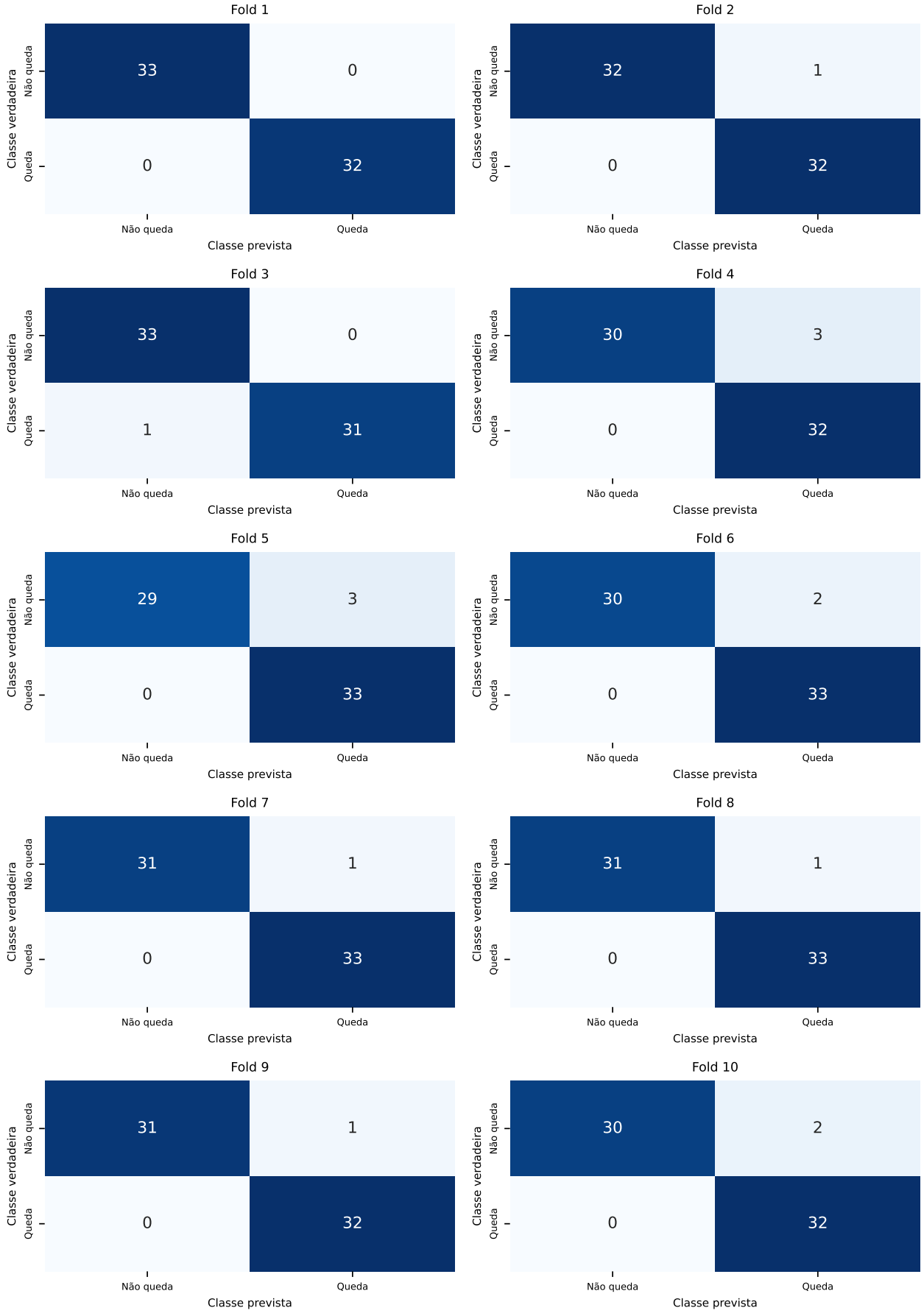


Figura 26: SVM: Matrizes de confusão obtidas para cada um dos 10 folds da validação cruzada, permitindo análise detalhada do desempenho preditivo por partição.