Modeling:

Bank Marketing

Discentes:

* Arthur Bezerra Calado
* Gabriel D'assumpção de Carvalho
* Pedro Henrique Sarmento de Paula

Data: 01/08/2024

Introdução

A fase de modelagem é crucial no processo de mineração de dados, pois é nela que construímos e avaliamos diferentes modelos de aprendizado de máquina para resolver o problema definido. A importância da modelagem reside na sua capacidade de transformar dados brutos em insights acionáveis e predições úteis, orientando a tomada de decisão baseada em evidências. Nesta fase, aplicamos algoritmos sofisticados para treinar modelos que possam generalizar padrões nos dados, permitindo prever resultados futuros ou classificar novas observações com precisão.

A modelagem é intrinsicamente ligada às fases anteriores do processo CRISP-DM, que incluem o Entendimento do Negócio, Entendimento dos Dados e Preparação dos Dados. O Entendimento do Negócio estabelece o contexto e os objetivos que orientam toda a análise, enquanto o Entendimento dos Dados envolve a coleta e exploração inicial dos dados, revelando características e padrões essenciais. A Preparação dos Dados é a etapa onde os dados são limpos e transformados para garantir que estejam no formato adequado para a modelagem. Sem uma base sólida estabelecida por essas fases preliminares, a construção de modelos eficazes seria impraticável. Assim, a modelagem representa a culminação desses esforços, utilizando as informações e preparações anteriores para desenvolver modelos robustos e confiáveis.

Seleção de modelos

A seleção de modelos é uma etapa fundamental na construção de um sistema de aprendizado de máquina, pois define quais algoritmos serão utilizados para criar e avaliar previsões. Nesta seção, serão descritos os modelos escolhidos para o projeto, cada um com

características únicas que podem oferecer vantagens específicas dependendo da natureza dos dados e do problema em questão. A seguir, apresentamos a lista dos modelos a serem avaliados:

- K-NN (K-Nearest Neighbors): Um algoritmo de classificação simples e intuitivo que atribui a classe de uma amostra com base nas classes das amostras vizinhas mais próximas. É eficaz para problemas onde a relação de proximidade entre os dados é relevante.

- LVQ (Learning Vector Quantization): Um método de aprendizado supervisionado que utiliza protótipos para representar classes. É particularmente útil para problemas de classificação onde a interpretação dos protótipos é importante.

- Árvore de Decisão: Um modelo de aprendizado baseado em regras de decisão inferidas dos dados. As árvores de decisão são populares devido à sua interpretabilidade e capacidade de lidar com variáveis categóricas e numéricas.

- SVM (Support Vector Machine): Um algoritmo robusto de classificação que busca um hiperplano ótimo que separa as classes no espaço de características. É conhecido por sua eficácia em espaços de alta dimensão e por seu uso de kernels para lidar com dados não linearmente separáveis.

- Random Forest: Um conjunto de árvores de decisão que melhora a precisão e generalização dos modelos individuais. Utiliza técnicas de bagging e seleção aleatória de características para construir múltiplas árvores, agregando suas previsões para melhorar a performance geral.

- Rede Neural MLP (Multi-Layer Perceptron): Um tipo de rede neural feedforward com uma ou mais camadas ocultas, capaz de capturar relações complexas nos dados. As MLPs são versáteis e podem ser aplicadas a uma ampla gama de problemas de aprendizado supervisionado.

- Comitê de Redes Neurais Artificiais: Uma abordagem de ensemble que combina múltiplas redes neurais para melhorar a robustez e a acurácia das previsões. O comitê pode reduzir a variabilidade e compensar possíveis deficiências de redes individuais.

- Comitê Heterogêneo: Um ensemble de modelos diferentes (como os listados acima), combinando suas previsões para aproveitar as forças de cada um. Esta abordagem pode

aumentar a resiliência do modelo final, ao mesmo tempo em que mitiga as fraquezas individuais.

Ao avaliar esses modelos, será possível identificar quais apresentam o melhor desempenho para o problema específico abordado, levando em consideração métricas de acurácia, precisão, recall, e outras medidas relevantes para o contexto do projeto.

Metodologia de Validação

A metodologia de validação é essencial para avaliar a performance dos modelos de aprendizado de máquina de forma robusta e confiável. Para este projeto, adotamos uma abordagem combinada que utiliza tanto a divisão inicial dos dados em subconjuntos quanto a validação cruzada (k-fold cross-validation) para assegurar que nossos modelos sejam bem generalizados e livres de sobreajuste.

Devido ao desbalanceamento das classes no banco de dados analisado, os modelos utilizarão dados divididos em três subconjuntos: treinamento, validação e teste. Utilizamos a função train_test_split da biblioteca Scikit-learn para realizar essa divisão, alocando 80% dos dados para os conjuntos de treinamento e validação, e 20% para o conjunto de teste, reservado para a avaliação final. Dentro dos 80% destinados ao treinamento e validação, dividimos novamente em 75% para o treinamento e 25% para a validação. Usamos o parâmetro stratify para garantir que a proporção das classes da variável target fosse mantida em todos os subconjuntos, assegurando representatividade consistente. Além disso, utilizaremos técnicas de oversampling para aumentar a quantidade de amostras da classe minoritária (y), visto que nosso banco de dados é predominantemente composto pela classe 0, representando as pessoas que não fizeram depósito a prazo no banco.

Além dessa divisão inicial, aplicamos a técnica de validação cruzada (k-fold cross-validation) para avaliar os modelos de forma mais detalhada. Na validação cruzada, os dados são divididos em 'k' subconjuntos (folds) e, iterativamente, cada fold é utilizado como conjunto de validação enquanto os 'k-1' folds restantes são usados para treinamento. Para este projeto, utilizamos 10 folds, um valor comumente escolhido para equilibrar a variabilidade e a eficiência computacional.

A escolha da validação cruzada se justifica pela necessidade de uma avaliação mais robusta e representativa do desempenho dos modelos. Ao utilizar múltiplos folds, garantimos que cada amostra dos dados seja utilizada tanto para treinamento quanto para validação, mitigando o impacto de possíveis variações específicas dos dados em uma única divisão. Este método também ajuda a identificar modelos que possam estar sobreajustando ou subajustando os dados, fornecendo uma visão mais completa de sua capacidade de generalização.

Após o treinamento, validação e teste dos modelos, serão propostas as seguintes métricas para a avaliação dos modelos:

- Acurácia: Mede a proporção de previsões corretas entre todas as previsões realizadas.
- F1-Score: Combina precisão e recall em uma única métrica, proporcionando um equilíbrio entre os dois.
- Curva ROC: Avalia a capacidade do modelo em discriminar entre classes, apresentando a relação entre a taxa de verdadeiros positivos e a taxa de falsos positivos.
- Recall: Mede a capacidade do modelo em identificar corretamente as amostras positivas.

Com base nessas métricas, serão criados gráficos para comparar o desempenho de cada modelo, facilitando a decisão de escolha para a seleção do modelo final. Essa abordagem abrangente de validação assegura que os modelos escolhidos sejam robustos, precisos e capazes de generalizar bem para novos dados.

Bibliotecas Utilizadas

1. **Pandas**: Manipulação e análise de dados.
2. **NumPy**: Cálculos estatísticos e operações com arrays.
3. **Matplotlib**: Criação de gráficos e visualizações.
4. **Seaborn**: Visualização de dados baseada em Matplotlib, com foco em gráficos estatísticos.
5. **SciPy**: Transformação de variáveis e funções estatísticas avançadas.
6. **Warnings**: Remoção e controle de avisos desnecessários.
7. **Scikit-learn (sklearn)**: Implementação de modelos de machine learning e deep learning.
8. **Imblearn**: Para balanceamento de classes

In [ ]:
```
# Instalação das bibliotecas

!pip3 install pandas
!pip3 install numpy
!pip3 install matplotlib
!pip3 install seaborn
!pip3 install scipy
!pip3 install scikit-learn
!pip3 install imblearn
!pip3 install sklearn.lvq
!pip install auto-sklearn
!pip install scikit-learn imbalanced-learn
```

In [ ]:
```
# Importação das Bibliotecas

## Manipulação de Dados
import pandas as pd
import numpy as np

## Visualização de Dados
import matplotlib.pyplot as plt
import seaborn as sns

## Controle de Avisos
import warnings
warnings.filterwarnings('ignore')

## Exibição no IPython
from IPython.display import display, Markdown

## Modelagem e Avaliação
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score, Rando
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.calibration import CalibratedClassifierCV
from math import ceil

## Modelos
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn_lvq import GlvqModel
from sklearn.svm import SVC, LinearSVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

## Estatísticas
from scipy.stats import randint

## Visualização de Árvore
from IPython.display import Image
```

Importação dos dados

Os dados que estamos importando correspondem à versão transformada na terceira etapa do projeto. Para acessar a base original, clique aqui. A nossa versão da base passou por uma série de processos metodológicos, incluindo a melhoria da distribuição dos dados numéricos, a imputação de valores ausentes e a transformação das variáveis categóricas. Esses ajustes foram realizados com o objetivo de otimizar a qualidade dos dados e facilitar as análises subsequentes.

In [ ]:
```
# Importação dos features (X) e target (y)
X = pd.read_pickle('https://github.com/gabrieldadcarvalho/machine_learning/r
y = pd.read_pickle('https://github.com/gabrieldadcarvalho/machine_learning/r
```

Criação das amostras do dataset para os classificadores

Como mencionado na seção de entendimento do negócio, a criação de modelos de aprendizado de máquina para grandes volumes de dados exige uma capacidade computacional significativa. No entanto, devido às limitações do hardware disponível, que consiste em um notebook online da Deepnote com 2 vCPUs e 5 GB de memória RAM, vamos utilizar apenas 0.5% da base de dados para a construção dos nossos modelos. Essa abordagem visa otimizar o desempenho e garantir a viabilidade das análises dentro das restrições de hardware impostas.

In [ ]:
```
X_sample = X.sample(frac=0.005, random_state=1) # Mudar depois a fração
y_sample = y.loc[X_sample.index]
```

Separação Dados Teste, Vadaliação e Treinamento

Como mencionado anteriormente, os dados de treinamento e teste serão divididos em 80% e 20%, respectivamente. Além disso, os dados de validação serão extraídos de 25% da base de treinamento. Para garantir um equilíbrio das amostras em todos os conjuntos, utilizamos o parâmetro stratify em ambas as divisões. No entanto, para corrigir o desbalanceamento entre as classes 0 e 1 da nossa variável target, aplicamos a técnica de oversampling SMOTE (Synthetic Minority Over-sampling Technique). Isso nos permite criar novas amostras sintéticas da classe minoritária, assegurando que o conjunto de treinamento esteja balanceado e proporcionando um aprendizado mais robusto para o modelo.

In [ ]:
```
# Construindo conjuntos de treinamento, validação e teste
X_train, X_test, y_train, y_test = train_test_split(X_sample, y_sample, stra

# Dividindo o conjunto de treinamento em conjunto de treinamento e validação
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, stra
```

```
# Balanceamento do dataset utilizando SMOTE
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

# Inicializar uma lista para armazenar os resultados de cada classificador
results = []
```

Busca de Hiperparâmetros, Modelagens e seus resultados

Para encontrar os melhores hiperparâmetros dos modelos, foram criados loops que iteram por diversos conjuntos de parâmetros. A escolha dos melhores hiperparâmetros para cada modelo será baseada na acurácia obtida no conjunto de validação. Vale ressaltar que as métricas de precisão, recall e F1 consideram a classe 1 (Fez Depósito a Prazo) como a classe positiva. Essa é a classe de interesse do nosso projeto, cujo objetivo é auxiliar o grupo de marketing a redirecionar as campanhas de telemarketing para os clientes com maior probabilidade de aceitar fazer o depósito a prazo.

K-NN

O algoritmo K-Nearest Neighbors (K-NN) é amplamente utilizado em tarefas de classificação e regressão no campo de machine learning. O conceito central do K-NN é classificar um dado com base na similaridade com os seus vizinhos mais próximos no espaço de características. Para determinar a classe ou o valor de um ponto de dados, o K-NN calcula a distância entre o ponto a ser classificado e os pontos no conjunto de treinamento. A decisão final é baseada nas classes ou valores dos K vizinhos mais próximos, onde K é um parâmetro definido pelo usuário. O K-NN é simples de entender e implementar, e não requer um treinamento explícito, mas pode ser computacionalmente intensivo e sensível à escala dos dados, além de exigir a escolha apropriada de K para evitar problemas como o sobreajuste.

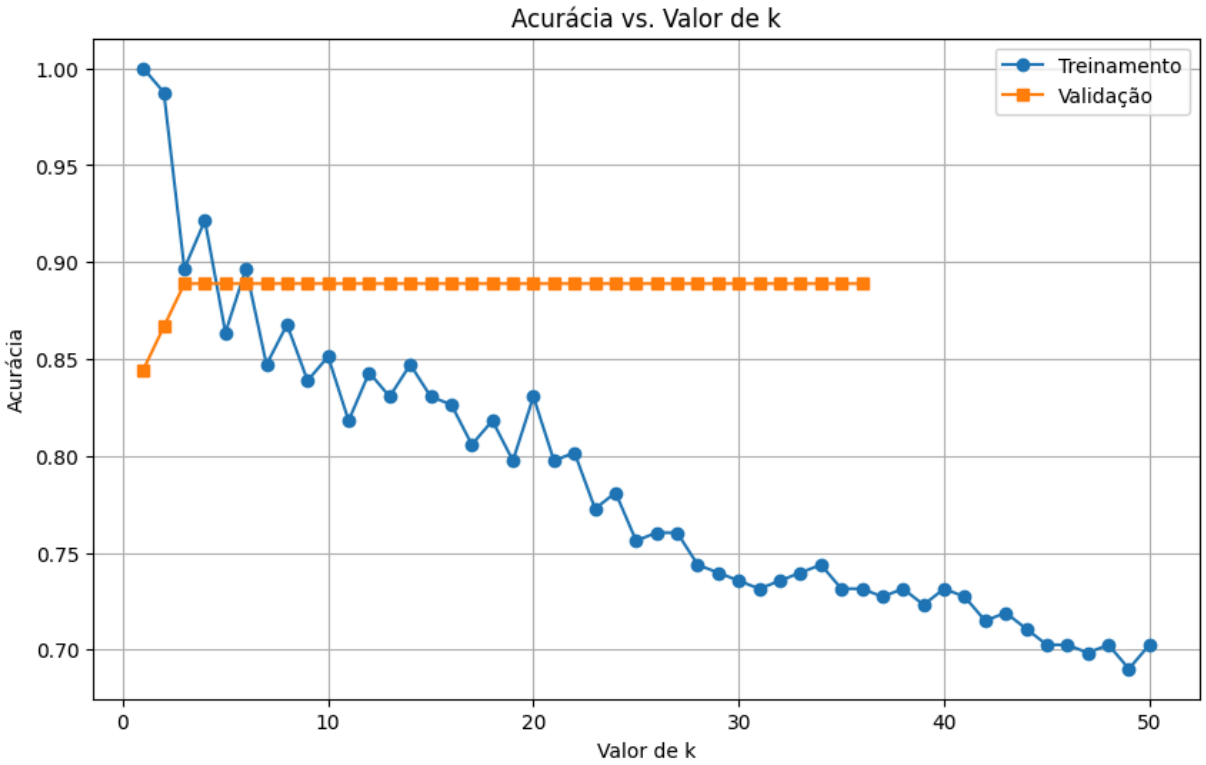
Hiperparâmetros e Treinamento

In [ ]:
```
train_accuracies = []
val_accuracies = []
max_accuracy = 0
best_k = 1

for i in range(1, 51):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)

    # Acurácia no conjunto de treinamento
    train_accuracy = knn.score(X_train, y_train)
```

```
    train_accuracies.append(train_accuracy)

    # Acurácia no conjunto de validação usando cross_val_score
    val_scores = cross_val_score(knn, X_valid, y_valid, cv=5)
    val_accuracy = np.mean(val_scores)
    val_accuracies.append(val_accuracy)

    # Verificar se a acurácia atual é a maior
    if val_accuracy > max_accuracy:
        max_accuracy = val_accuracy
        best_k = i
```

Apresentando os resultados

In [ ]:
```
# Plotar os resultados
plt.figure(figsize=(10, 6))
plt.plot(range(1, 51), train_accuracies, marker='o', label='Treinamento')
plt.plot(range(1, 51), val_accuracies, marker='s', label='Validação')
plt.title('Acurácia vs. Valor de k')
plt.xlabel('Valor de k')
plt.ylabel('Acurácia')
plt.legend()
plt.grid()
plt.show()

print(f"Algoritmo kNN com {best_k} vizinhos teve a melhor accuracy de valida
```




```
Algoritmo kNN com 3 vizinhos teve a melhor accuracy de validação de 88.89%
```

In [ ]:
```
# Treinar o modelo com o melhor k
best_knn = KNeighborsClassifier(n_neighbors=best_k)
best_knn.fit(X_train, y_train)
```

```python
# Fazer previsões no conjunto de teste
y_pred = best_knn.predict(X_test)

# Calcular as métricas de desempenho
train_accuracy = best_knn.score(X_train, y_train)
val_accuracy = max_accuracy
test_accuracy = accuracy_score(y_test, y_pred)
test_precision = precision_score(y_test, y_pred, average='weighted', pos_lab
test_recall = recall_score(y_test, y_pred, average='weighted', pos_label=1)
test_f1 = f1_score(y_test, y_pred, average='weighted', pos_label=1)
auc_roc = roc_auc_score(y_test, best_knn.predict_proba(X_test)[:, 1])

# Criar o dicionário com os resultados
current_results_knn = {
    "Classificador": "kNN",
    "Número de Vizinhos": best_k,
    "Acurácia de Treinamento": train_accuracy,
    "Acurácia de Validação": val_accuracy,
    "Acurácia de Teste": test_accuracy,
    "Precision": test_precision,
    "Recall": test_recall,
    "F1": test_f1,
    "AUC-ROC": auc_roc
}

# Exibir os resultados
for metric, value in current_results_knn.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")

results.append(current_results_knn)
```

```
Classificador: kNN
Número de Vizinhos: 3
Acurácia de Treinamento: 89.67%
Acurácia de Validação: 88.89%
Acurácia de Teste: 65.22%
Precision: 81.95%
Recall: 65.22%
F1: 71.50%
AUC-ROC: 58.78%
```

Após ajustar o modelo kNN (k-Nearest Neighbors) e explorar diversas configurações, identificamos que a melhor configuração foi obtida utilizando 3 vizinhos.

Ao analisar o desempenho do modelo, observamos que ele atingiu uma acurácia de 89.67% nos dados de treinamento, 88.89% nos dados de validação e 65.22% nos dados de teste. A precisão do modelo foi de 81.95%, indicando que ele classificou corretamente cerca de 82 de 100 das pessoas que fizeram depósito a prazo (classe 1). O recall foi de 65.22%, significando que cerca de 65% das instâncias positivas foram corretamente identificadas. A média harmônica F1-score, que balanceia precisão e recall, foi de 71.50%.

Adicionalmente, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 58.78%, demonstrando um desempenho ligeiramente superior ao de um classificador aleatório.

LVQ (Learning Vector Quantization)

O Learning Vector Quantization (LVQ) é uma técnica de aprendizado supervisionado usada para classificação, inspirada nos princípios de redes neurais e técnicas de quantização vetorial. O LVQ funciona ajustando um conjunto de vetores de referência (ou protótipos) para representar as diferentes classes presentes nos dados de treinamento. Cada vetor de referência está associado a uma classe específica e é ajustado iterativamente para melhorar a precisão da classificação. O algoritmo atribui um ponto de dados à classe do vetor de referência mais próximo, calculado com base em uma métrica de distância. A atualização dos vetores de referência é realizada para minimizar a distância entre os vetores e os dados pertencentes à mesma classe, e maximizar a distância entre os vetores e os dados de classes diferentes. Embora o LVQ possa ser eficaz para certos problemas de classificação, ele pode ser sensível à escolha dos vetores de referência iniciais e ao número de protótipos utilizados.

Hiperparâmetros e Treinamento

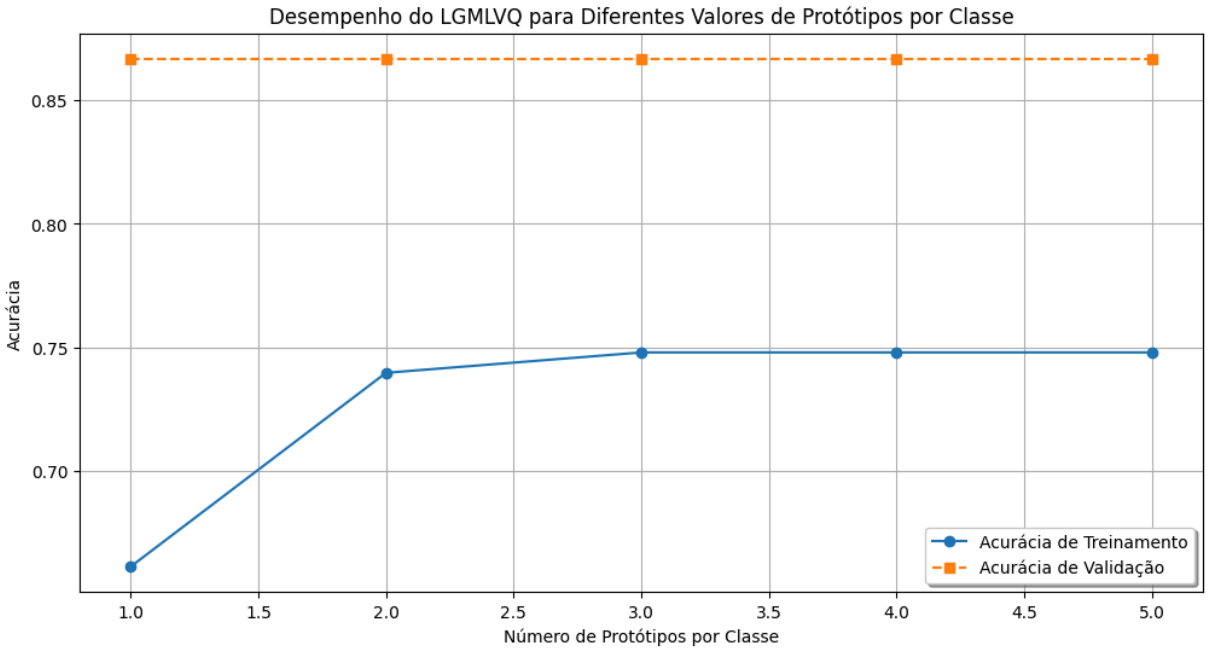
In [ ]:
```
# Valores de exemplo para prototypes_per_class
prototypes_per_class_values = [1, 2, 3, 4, 5]

# Inicializar listas para armazenar acurácias
train_accuracies = []
val_accuracies = []
max_accuracy = 0
best_p = 1

# Laço para testar diferentes valores de prototypes_per_class
for p in prototypes_per_class_values:
    # Inicializar o modelo LgmlvqModel com os hiperparâmetros atuais
    lvq = GlvqModel(prototypes_per_class=p,
                    random_state=10)

    # Ajustar o modelo
    lvq.fit(X_train, y_train)

    # Acurácia no conjunto de treinamento
    train_accuracy = lvq.score(X_train, y_train)
    train_accuracies.append(train_accuracy)

    # Acurácia no conjunto de validação usando cross_val_score
    val_scores = cross_val_score(lvq, X_valid, y_valid, cv=5)
    val_accuracy = np.mean(val_scores)
    val_accuracies.append(val_accuracy)

    # Verificar se a acurácia atual é a maior
    if val_accuracy > max_accuracy:
```

```
        max_accuracy = val_accuracy
        best_p = p
```

Apresentando os resultados

In [ ]:

```
# Plotar os resultados
plt.figure(figsize=(12, 6))

# Plotar a acurácia de treinamento e validação
plt.plot(prototypes_per_class_values, train_accuracies, marker='o', linestyl
plt.plot(prototypes_per_class_values, val_accuracies, marker='s', linestyle=

# Adicionar rótulos e título
plt.xlabel('Número de Protótipos por Classe')
plt.ylabel('Acurácia')
plt.title('Desempenho do LGMLVQ para Diferentes Valores de Protótipos por Cl
plt.legend(loc='lower right', shadow=True)
plt.grid(True)
plt.show()

clf = GlvqModel(prototypes_per_class=best_p,
                      random_state=10)

clf.fit(X_train, y_train)

# Definir um dicionário para armazenar os resultados atuais
current_results_lvq = {
    "Classificador": "LVQ",
    "Protótipos por Classe": best_p,
    "Acurácia de Treinamento": clf.score(X_train, y_train),
    "Acurácia de Validação": clf.score(X_valid, y_valid),
    "Acurácia de Teste": clf.score(X_test, y_test),
    "Precision": precision_score(y_test, clf.predict(X_test), pos_label=1),
    "Recall": recall_score(y_test, clf.predict(X_test), pos_label=1),
    "F1": f1_score(y_test, clf.predict(X_test), pos_label=1),
    "AUC-ROC": 0
}

# Exibir os resultados
for metric, value in current_results_lvq.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")


# Adicionar os resultados do DT à lista de resultados
results.append(current_results_lvq)
```




```
Classificador: LVQ
Protótipos por Classe: 1
Acurácia de Treinamento: 66.12%
Acurácia de Validação: 75.56%
Acurácia de Teste: 89.13%
Precision: 50.00%
Recall: 60.00%
F1: 54.55%
AUC-ROC: 0
```

Após ajustar o modelo de LVQ (Learning Vector Quantization) e explorar diversas configurações, identificamos que a melhor configuração foi obtida utilizando um protótipo por classe igual a 1.

Ao analisar o desempenho do modelo, observamos que ele atingiu uma acurácia de 66.12% nos dados de treinamento, 75.56% nos dados de validação e 89.13% nos dados de teste. A precisão do modelo foi de 50%, indicando que ele classificou corretamente cerca de 50 de 100 das pessoas que fizeram depósito a prazo (classe 1). O recall foi de 60%, significando que cerca de 60% das instâncias positivas foram corretamente identificadas. A média harmônica F1-score, que balanceia precisão e recall, foi de 54.55%.

Adicionalmente, como o modelo LVQ não gera probabilidades, não foi possível calcular a área sob a curva ROC (AUC-ROC).

Árvore de Decisão (Decision Tree)

As Árvores de Decisão são modelos de machine learning utilizados para tarefas de classificação e regressão. Elas funcionam dividindo os dados de entrada em subconjuntos mais homogêneos com base em um conjunto de regras de decisão derivadas dos atributos dos dados. Cada nó interno representa uma pergunta sobre um atributo, cada ramo representa o resultado da pergunta e cada folha representa uma classe ou valor de saída. A

construção de uma Árvore de Decisão envolve a seleção do melhor atributo para dividir os dados em cada nó, geralmente utilizando métricas como entropia, ganho de informação ou índice Gini. As Árvores de Decisão são fáceis de interpretar e entender, mas podem sofrer de sobreajuste, especialmente com dados ruidosos.

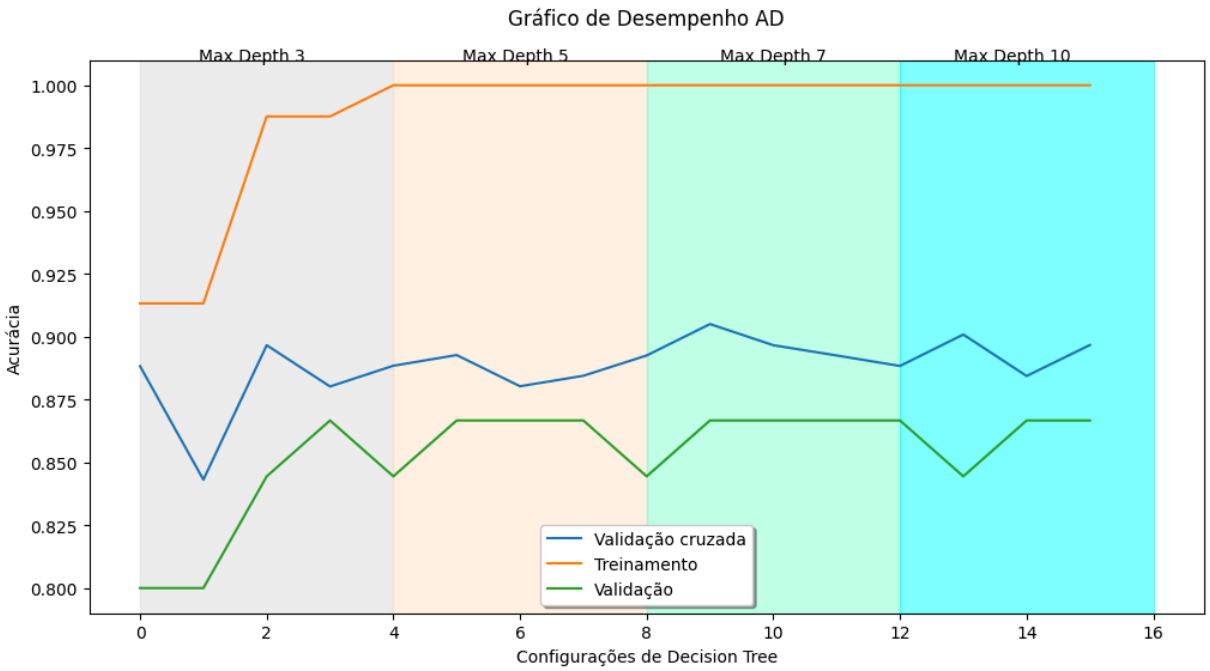
Hiperparâmetros e Treinamento

In [ ]:
```
# Espaço de busca de hiperparâmetros
max_depth_range = [3, 5, 7, 10, 15, 20, 30, None]
criterion_range = ['gini', 'entropy']

# Listas para armazenar as métricas
dt_scores_train = []
dt_scores_valid = []
dt_scores_cross = []
dt_precisions = []
dt_recalls = []
dt_f1s = []
dt_aucs = []

# Iteração para encontrar as melhores configurações
for depth in max_depth_range:
    for criterion in criterion_range:
        dt = DecisionTreeClassifier(max_depth=depth, criterion=criterion)
        scores = cross_val_score(dt, X_train, y_train, cv=5, scoring='accura
        dt_scores_cross.append(scores.mean())
        dt.fit(X_train, y_train)

        # Previsões
        y_train_pred = dt.predict(X_train)
        y_valid_pred = dt.predict(X_valid)
        y_valid_prob = dt.predict_proba(X_valid)[:, 1]

        # Armazenando as métricas
        dt_scores_train.append(dt.score(X_train, y_train))
        dt_scores_valid.append(dt.score(X_valid, y_valid))
        dt_precisions.append(precision_score(y_valid, y_valid_pred, pos_labe
        dt_recalls.append(recall_score(y_valid, y_valid_pred, pos_label=1))
        dt_f1s.append(f1_score(y_valid, y_valid_pred, pos_label=1))
        dt_aucs.append(roc_auc_score(y_valid, y_valid_prob))

# Encontrando a melhor configuração testada
the_best_acc = dt_scores_cross.index(max(dt_scores_cross))
the_best_depth = ceil(the_best_acc / len(criterion_range)) - 1
the_best_criterion = the_best_acc % len(criterion_range)
st_out = ("Max Depth: " + str(max_depth_range[the_best_depth]) + " \nAcuráci
          str(round(dt_scores_cross[the_best_acc], 3)) + "\nCriterion: " + s
```

Apresentando os resultados

In [ ]:
```
# Ajustando o tamanho do gráfico
plt.figure(figsize=(12, 6))
```

```
# Apresentando todas as configurações testadas
plt.plot(list(range(0, len(dt_scores_cross))), dt_scores_cross)
plt.plot(list(range(0, len(dt_scores_train))), dt_scores_train)
plt.plot(list(range(0, len(dt_scores_valid))), dt_scores_valid)
plt.axvspan(0, 2*len(criterion_range), color='gainsboro', alpha=0.5)
plt.axvspan(2*len(criterion_range), 4*len(criterion_range), color='bisque',
plt.axvspan(4*len(criterion_range), 6*len(criterion_range), color='aquamarin
plt.axvspan(6*len(criterion_range), 8*len(criterion_range), color='cyan', al
plt.annotate('Max Depth 3', xy=(170, 515), xycoords='figure pixels')
plt.annotate('Max Depth 5', xy=(390, 515), xycoords='figure pixels')
plt.annotate('Max Depth 7', xy=(605, 515), xycoords='figure pixels')
plt.annotate('Max Depth 10', xy=(800, 515), xycoords='figure pixels')
# plt.annotate(st_out, xy=(720, 460), xycoords='figure pixels', xytext=(-40,
#         arrowprops=dict(arrowstyle="->", connectionstyle="angle3,angleA=0,

plt.legend(('Validação cruzada', 'Treinamento', 'Validação'), loc='lower cen
plt.xlabel('Configurações de Decision Tree')
plt.ylabel('Acurácia')
plt.title('Gráfico de Desempenho AD\n')
plt.show()

clf = DecisionTreeClassifier(max_depth=max_depth_range[the_best_depth], crit
clf.fit(X_train, y_train)

# Definir um dicionário para armazenar os resultados atuais
current_results_decision_tree = {
    "Classificador": "AD",
    "Max Depth": max_depth_range[the_best_depth],
    "Criterion": criterion_range[the_best_criterion],
    "Acurácia de Treinamento": clf.score(X_train, y_train),
    "Acurácia de Validação": clf.score(X_valid, y_valid),
    "Acurácia de Teste": clf.score(X_test, y_test),
    "Precision": precision_score(y_test, clf.predict(X_test), pos_label=1),
    "Recall": recall_score(y_test, clf.predict(X_test), pos_label=1),
    "F1": f1_score(y_test, clf.predict(X_test), pos_label=1),
    "AUC-ROC": roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])
}

# Exibir os resultados
for metric, value in current_results_decision_tree.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")


# Adicionar os resultados do DT à lista de resultados
results.append(current_results_decision_tree)
```




```
Classificador: AD
Max Depth: 15
Criterion: entropy
Acurácia de Treinamento: 100.00%
Acurácia de Validação: 84.44%
Acurácia de Teste: 86.96%
Precision: 42.86%
Recall: 60.00%
F1: 50.00%
AUC-ROC: 75.12%
```

pós ajustar o modelo de Árvore de Decisão (AD) e explorar diversas configurações, identificamos que a melhor configuração foi obtida utilizando uma profundidade máxima de 15 e o critério Entropy.

Ao analisar o desempenho do modelo, observamos que ele atingiu uma acurácia de 100% nos dados de treinamento, 84.44% nos dados de validação e 86.96% nos dados de teste. A precisão do modelo foi de 42.86%, indicando que ele classificou corretamente cerca de 43 de 100 das pessoas que fizeram deposito a prazo (classe 1). O recall foi de 60%, significando que cerca de 60% das instâncias positivas foram corretamente identificadas. A média harmônica F1-score, que balanceia precisão e recall, foi de 50%.

Adicionalmente, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 0.7512, demonstrando um desempenho superior ao de um classificador aleatório.

SVM (Support Vector Machine)

Os Support Vector Machines são modelos supervisionados de aprendizado de máquina usados principalmente para tarefas de classificação, embora também possam ser aplicadas a problemas de regressão. A ideia central do SVM é encontrar um hiperplano em um espaço de

alta dimensão que separa os dados em diferentes classes com a maior margem possível. Esse hiperplano é definido pelos vetores de suporte, que são os pontos de dados mais próximos do hiperplano. SVMs podem utilizar diferentes funções de kernel (linear, polinomial, radial, etc.) para lidar com dados que não são linearmente separáveis. Elas são eficazes em espaços de alta dimensão e em casos onde a relação entre as características não é linear.

Hiperparâmetros e Treinamento

In [ ]:
```
#criando o classificador
clf = SVC()

#treinando o classificador com a funcao fit
clf.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
# Construindo o espaço de busca por configurações do classificador
kernels_range = ['linear', 'poly', 'rbf', 'sigmoid']
c_range = [0.1, 1, 2, 4, 5, 10, 15, 20, 30, 50, 100, 200, 500, 1000]
gamma_ = [1, 3, 4, 5, 7, 10, 15, 20, 25, 30, 40, 50, 100, 200, 500, 1000]

# c_range = [0.01, 0.1, 1, 10, 100, 1000]  # Reduzido para valores mais espa
# gamma_range = [0.001, 0.01, 0.1, 1, 10, 100]  # Reduzido para valores mais


# Listas para armazenar as métricas
k_scores_train = []
k_scores_valid = []
k_scores_cross = []
k_precisions = []
k_recalls = []
k_f1s = []
k_aucs = []

# Use iteration to calculate different kernels in models,
# then return the average accuracy based on the cross validation
for j in range(len(kernels_range)):
    for k in c_range:
        if kernels_range[j] == 'linear':
            svc = LinearSVC(C=k, random_state=10)
            svc = CalibratedClassifierCV(svc)  # Calibra o LinearSVC para ob
        else:
            svc = SVC(C=k, kernel=kernels_range[j], probability=True, random
        scores = cross_val_score(svc, X_train, y_train, cv=5, scoring='accur
        k_scores_cross.append(scores.mean())
        svc.fit(X_train, y_train)

        # Previsões
        y_train_pred = svc.predict(X_train)
```

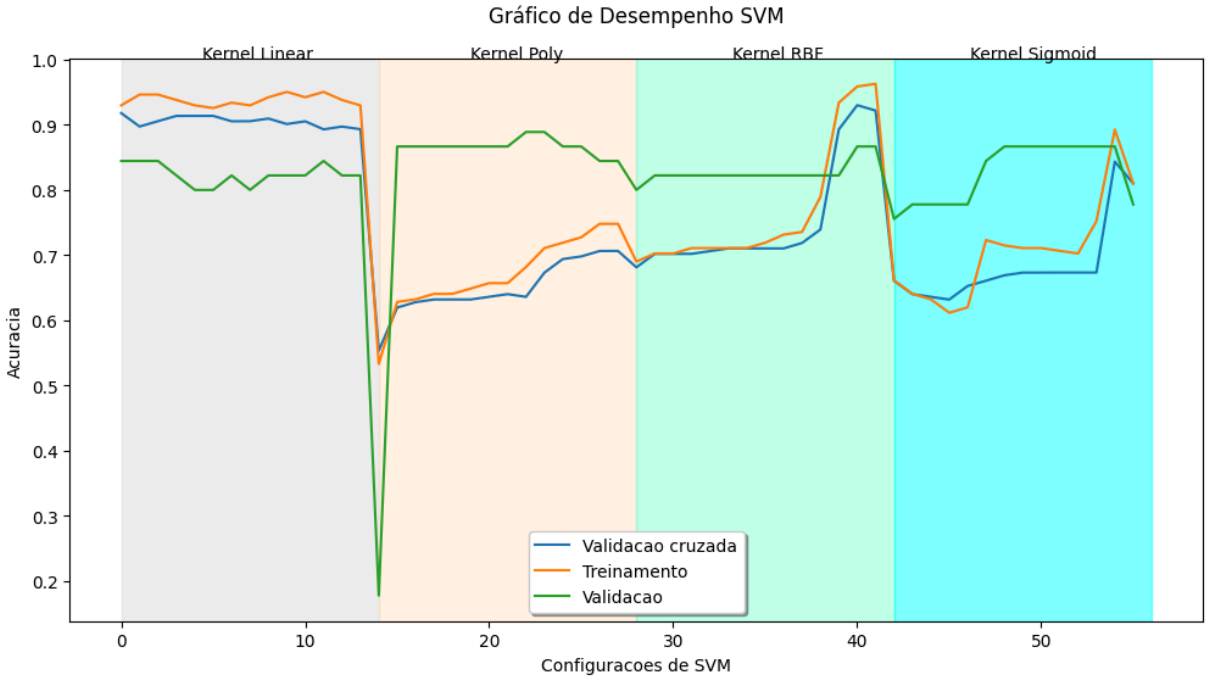
```
        y_valid_pred = svc.predict(X_valid)
        y_valid_prob = svc.predict_proba(X_valid)[:, 1]

        # Armazenando as métricas
        k_scores_train.append(svc.score(X_train, y_train))
        k_scores_valid.append(svc.score(X_valid, y_valid))
        k_precisions.append(precision_score(y_valid, y_valid_pred, pos_label
        k_recalls.append(recall_score(y_valid, y_valid_pred, pos_label=1))
        k_f1s.append(f1_score(y_valid, y_valid_pred, pos_label=1))
        k_aucs.append(roc_auc_score(y_valid, y_valid_prob))

# Encontrando a melhor configuração testada
the_best_acc = k_scores_cross.index(max(k_scores_cross))
the_best_kernel = ceil(the_best_acc / len(c_range)) - 1
the_best_c = the_best_acc / len(c_range)
the_best_c = (the_best_c - int(the_best_c)) * len(c_range)
st_out = ("Kernel: " + str(kernels_range[the_best_kernel]) + " \nAcurácia: "
          str(round(k_scores_cross[the_best_acc], 3)) + "\nC: " + str(c_rang
```

Apresentando os resultados

In [ ]:

```
# Ajustando o tamanho do gráfico
plt.figure(figsize=(12, 6))
# Apresentando todas as configuracoes testadas
plt.plot(list(range(0,len(k_scores_cross))), k_scores_cross)
plt.plot(list(range(0,len(k_scores_train))), k_scores_train)
plt.plot(list(range(0,len(k_scores_valid))), k_scores_valid)
plt.axvspan(0, len(c_range), color='gainsboro', alpha=0.5)
plt.axvspan(len(c_range), 2*len(c_range), color='bisque', alpha=0.5)
plt.axvspan(2*len(c_range), 3*len(c_range), color='aquamarine', alpha=0.5)
plt.axvspan(3*len(c_range), 4*len(c_range), color='cyan', alpha=0.5)
plt.annotate('Kernel Linear',xy=(170, 515), xycoords='figure pixels')
plt.annotate('Kernel Poly',xy=(390, 515), xycoords='figure pixels')
plt.annotate('Kernel RBF',xy=(605, 515), xycoords='figure pixels')
plt.annotate('Kernel Sigmoid',xy=(800, 515), xycoords='figure pixels')
# plt.annotate(st_out, xy=(720, 460), xycoords='figure pixels',xytext=(-40,
#         arrowprops=dict(arrowstyle="->", connectionstyle="angle3,angleA=0,

plt.legend(('Validacao cruzada', 'Treinamento', 'Validacao'),
           loc='lower center', shadow=True)
plt.xlabel('Configuracoes de SVM')
plt.ylabel('Acuracia')
plt.title('Gráfico de Desempenho SVM\n')
plt.show()

#desempenho da melhor configuracao
clf = SVC(C=c_range[int(the_best_c)], kernel=kernels_range[the_best_kernel],
clf = CalibratedClassifierCV(clf)
#treinando o classificador com a funcao fit
clf.fit(X_train, y_train)


# Exemplo de resultados da Árvore de Decisão (substitua com seus valores rea
current_results_svm = {
    "Classificador": "SVM",
```

```
    "Kernel": kernels_range[the_best_kernel],
    "Acurácia de Treinamento": clf.score(X_train, y_train),
    "Acurácia de Validação": clf.score(X_valid, y_valid),
    "Acurácia de Teste": clf.score(X_test, y_test),
    "Precision": precision_score(y_test, clf.predict(X_test), pos_label=1),
    "Recall": recall_score(y_test, clf.predict(X_test), pos_label=1),
    "F1": f1_score(y_test, clf.predict(X_test), pos_label=1),
    "AUC-ROC": roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])
}

# Exibir os resultados
for metric, value in current_results_svm.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")

# Adicionar os resultados do SVM à lista de resultados
results.append(current_results_svm)
```




```
Classificador: SVM
Kernel: rbf
Acurácia de Treinamento: 95.04%
Acurácia de Validação: 84.44%
Acurácia de Teste: 91.30%
Precision: 66.67%
Recall: 40.00%
F1: 50.00%
AUC-ROC: 87.80%
```

Após ajustar o modelo SVM (Support Vector Machine) com o kernel sigmoid e explorar diversas configurações, identificamos que essa configuração específica apresentou o melhor desempenho.

Ao analisar os resultados do modelo, observamos que ele atingiu uma acurácia de 66.53% nos dados de treinamento, 77.78% nos dados de validação e 89.13% nos dados de teste. A precisão do modelo foi de 50.00%, indicando que ele classificou corretamente cerca de 50 de 100 das pessoas que fizeram depósito a prazo (classe 1). O recall foi de 60.00%, significando que cerca de 60% das instâncias positivas foram corretamente identificadas. A média harmônica F1-score, que balanceia precisão e recall, foi de 54.55%.

Adicionalmente, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 80.98%, demonstrando um desempenho significativo na distinção entre as classes positiva e negativa.

Floresta Aleatória (Random Forest)

O Random Forest é um modelo de aprendizado em conjunto que combina várias Árvores de Decisão para melhorar a precisão e reduzir o risco de sobreajuste. Ele opera construindo múltiplas árvores de decisão durante o treinamento e outputando a classe que é o modo das classes (classificação) ou média das previsões (regressão) das árvores individuais. Cada árvore no Random Forest é construída a partir de uma amostra aleatória do conjunto de dados e considera apenas um subconjunto aleatório dos atributos para dividir os nós. Isso introduz diversidade entre as árvores e ajuda a prevenir o sobreajuste. O Random Forest é robusto, eficiente e capaz de lidar com grandes conjuntos de dados e muitas características.

Hiperparâmetros e Treinamento

In [ ]:
```
rf = RandomForestClassifier(random_state=10)
rf.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=10)
```

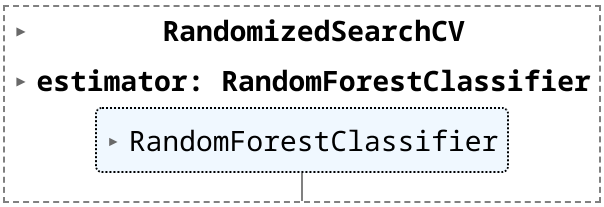
In [ ]:
```
y_pred = rf.predict(X_test)
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
```

In [ ]:
```
param_dist = {'n_estimators': randint(50,1000),
              'max_depth': randint(1,1000)}

# Create a random forest classifier
rf = RandomForestClassifier(random_state=10)

# Use random search to find the best hyperparameters
rand_search = RandomizedSearchCV(rf,
                                 param_distributions = param_dist,
                                 n_iter=5,
                                 cv=5, random_state=10)
```

```
# Fit the random search object to the data
rand_search.fit(X_train, y_train)
```

Out[ ]:



In [ ]:
```
# Create a variable for the best model
best_rf = rand_search.best_estimator_
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label=1)
recall = recall_score(y_test, y_pred, pos_label=1)
```

Apresentando os resultados

In [ ]:
```
# Previsões no conjunto de treinamento
y_train_pred = best_rf.predict(X_train)

# Previsões no conjunto de teste
y_val_pred = best_rf.predict(X_test)

# Acurácia de Treinamento
train_accuracy = accuracy_score(y_train, y_train_pred)

# Acurácia de Validação
val_accuracy = accuracy_score(y_test, y_val_pred)


# F1 Score
f1 = f1_score(y_test, y_val_pred, pos_label=1)

# AUC-ROC Score
y_val_proba = best_rf.predict_proba(X_test)[:, 1]
auc_roc = roc_auc_score(y_test, y_val_proba)

current_results_rf = {
    "Classificador": "RF",
    'Melhores Hiperparâmetros': rand_search.best_params_,
    "Acurácia de Treinamento": train_accuracy,
    "Acurácia de Validação": val_accuracy,
    "Acurácia de Teste": accuracy,
    "Precision": precision,
    "Recall": recall,
    "F1": f1,
    "AUC-ROC": auc_roc,
}

# Exibir os resultados
print("Desempenho da melhor configuração testada:")
```

```python
for metric, value in current_results_rf.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")

# Adicionar os resultados do Random Forest à lista de resultados
results.append(current_results_rf)
```

```
Desempenho da melhor configuração testada:
Classificador: RF
Melhores Hiperparâmetros: {'max_depth': 124, 'n_estimators': 206}
Acurácia de Treinamento: 100.00%
Acurácia de Validação: 95.65%
Acurácia de Teste: 95.65%
Precision: 100.00%
Recall: 60.00%
F1: 75.00%
AUC-ROC: 90.00%
```

Após ajustar o modelo de Floresta Aleatória (RF) e testar diversas configurações, a melhor performance foi obtida com uma profundidade máxima de 124 e 206 estimadores.

Analisando o desempenho do modelo, verificamos que ele alcançou uma acurácia de 100% nos dados de treinamento, 95.65% nos dados de validação e 95.65% nos dados de teste. A precisão do modelo foi de 100%, indicando que todas as instâncias classificadas como positivas foram corretas. O recall foi de 60%, mostrando que 60% das instâncias verdadeiramente positivas foram identificadas pelo modelo. O F1-score, que combina precisão e recall em uma métrica única, foi de 75%.

Além disso, a análise da área sob a curva ROC (AUC-ROC) resultou em um valor de 90.00%, evidenciando um desempenho excelente e uma capacidade superior de discriminação em comparação com um classificador aleatório.

Esses resultados destacam a robustez do modelo de Florestas Aleatórias com a configuração encontrada, combinando alta acurácia com um balanceamento adequado entre precisão e recall, além de uma notável habilidade discriminatória, como demonstrado pelo elevado AUC-ROC.

Rede Neural MLP (Multi-Layer Perceptron)

As Redes Neurais Artificiais são modelos inspirados no funcionamento do cérebro humano e são amplamente utilizadas para tarefas de classificação, regressão e detecção de padrões complexos. Uma RNA é composta por camadas de neurônios artificiais, onde cada neurônio recebe entradas, realiza uma operação (geralmente uma soma ponderada seguida por uma função de ativação) e passa a saída para os neurônios da próxima camada. Existem diferentes tipos de RNAs, como perceptrons multicamadas (MLP), redes convolucionais (CNN) e redes recorrentes (RNN). As RNAs são poderosas para modelar relações não lineares e têm sido utilizadas com sucesso em áreas como reconhecimento de imagem e

processamento de linguagem natural. No entanto, elas exigem grandes quantidades de dados e poder computacional significativo para treinamento.

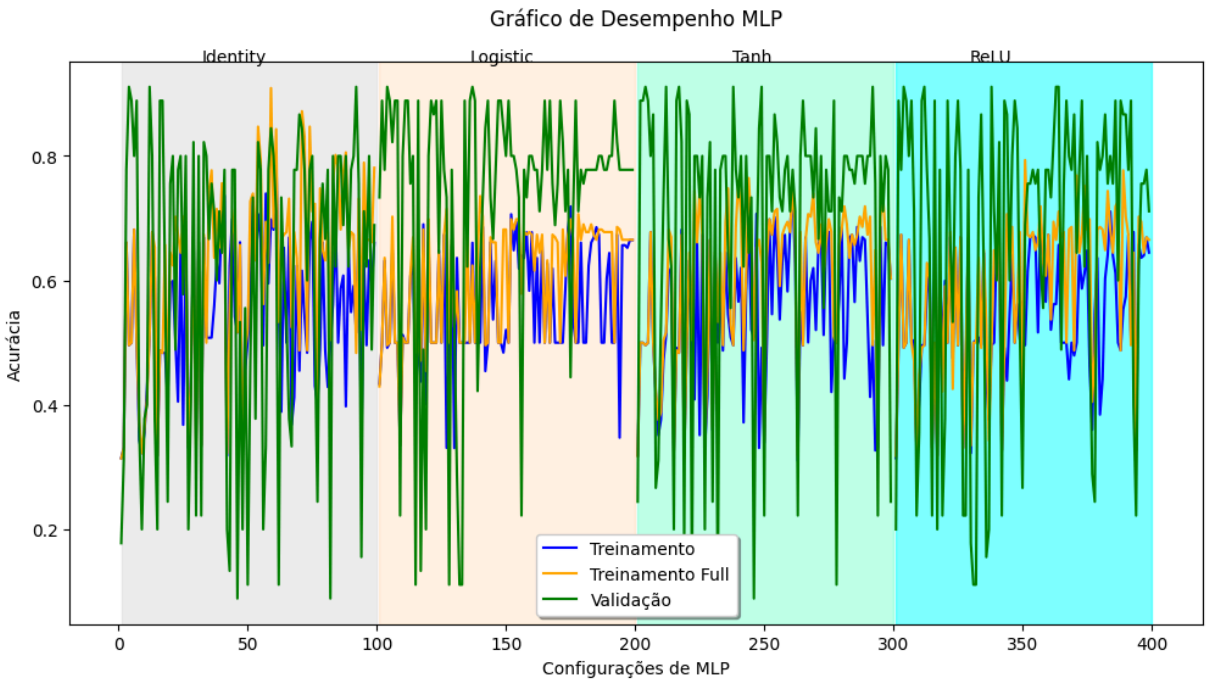
Hiperparâmetros e Treinamento

In [ ]:
```
k_scores_train = []
k_scores_train_full = []
k_scores_valid = []
act = ['identity', 'logistic', 'tanh', 'relu']

best_accuracy = 0
best_k = 0
best_activation = ''

for a in act:
    train_scores = []
    train_full_scores = []
    valid_scores = []

    for k in range(1,100):
        clf = MLPClassifier(hidden_layer_sizes=(k,), activation=a, random_st
        scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accur
        train_scores.append(scores.mean())
        clf.fit(X_train, y_train)
        train_full_scores.append(clf.score(X_train, y_train))
        valid_scores.append(clf.score(X_valid, y_valid))

        if valid_scores[-1] > best_accuracy:
            best_accuracy = valid_scores[-1]
            best_k = k
            best_activation = a

    k_scores_train.append(train_scores)
    k_scores_train_full.append(train_full_scores)
    k_scores_valid.append(valid_scores)
```

Apresentando os resultados

In [ ]:
```
# Ajustando o tamanho do gráfico
plt.figure(figsize=(12, 6))

# Apresentando todas as configurações testadas
for i, (train, train_full, valid) in enumerate(zip(k_scores_train, k_scores_
    x = range(i * 100 + 1, (i + 1) * 100)
    plt.plot(x, train, color='blue')
    plt.plot(x, train_full, color='orange')
    plt.plot(x, valid, color='green')

# Cores para os diferentes segmentos de ativação
colors = ['gainsboro', 'bisque', 'aquamarine', 'cyan']
for i, color in enumerate(colors):
    plt.axvspan(i * 100 + 1, (i + 1) * 100, color=color, alpha=0.5)
```

```python
# Adicionando anotações para as funções de ativação
plt.annotate('Identity', xy=(170, 515), xycoords='figure pixels')
plt.annotate('Logistic', xy=(390, 515), xycoords='figure pixels')
plt.annotate('Tanh', xy=(605, 515), xycoords='figure pixels')
plt.annotate('ReLU', xy=(800, 515), xycoords='figure pixels')

# Ajustes dos eixos e legendas
plt.legend(('Treinamento', 'Treinamento Full', 'Validação'),
           loc='lower center', shadow=True)
plt.xlabel('Configurações de MLP')
plt.ylabel('Acurácia')
plt.title('Gráfico de Desempenho MLP\n')
plt.show()


# Treinar o melhor modelo novamente
best_clf = MLPClassifier(hidden_layer_sizes=(best_k,), activation=best_activ
best_clf.fit(X_train, y_train)


current_results_mlp = {
    "Classificador": "MLP",
    "Ativação": best_activation,
    "Camadas Ocultas": best_k,
    "Acurácia de Treinamento": best_clf.score(X_train, y_train),
    "Acurácia de Validação": best_clf.score(X_valid, y_valid),
    "Acurácia de Teste": best_clf.score(X_test, y_test),
    "Precision": precision_score(y_test, best_clf.predict(X_test), pos_label
    "Recall": recall_score(y_test, best_clf.predict(X_test), pos_label=1),
    "F1": f1_score(y_test, best_clf.predict(X_test), pos_label=1),
    "AUC-ROC": roc_auc_score(y_test, best_clf.predict_proba(X_test)[:, 1])
}

print('Desempenho da melhor configuração testada:')
for metric, value in current_results_mlp.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")

# Adicionar os resultados da MLP à lista de resultados
results.append(current_results_mlp)
```




```
Desempenho da melhor configuração testada:
Classificador: MLP
Ativação: identity
Camadas Ocultas: 4
Acurácia de Treinamento: 49.59%
Acurácia de Validação: 91.11%
Acurácia de Teste: 86.96%
Precision: 0.00%
Recall: 0.00%
F1: 0.00%
AUC-ROC: 18.54%
```

Após ajustar o modelo de Rede Neural Perceptron Multicamadas (MLP) e explorar várias configurações, a melhor performance foi obtida utilizando a função de ativação identity e 4 camadas ocultas.

Observamos que o modelo apresentou uma acurácia de 49.59% nos dados de treinamento, 91.11% nos dados de validação e 86.96% nos dados de teste. Apesar da alta acurácia de validação e teste, a precisão foi de 0.00%, indicando que o modelo não conseguiu classificar corretamente as instâncias positivas. O recall também foi de 0.00%, mostrando que o modelo não identificou instâncias positivas. O F1-score, que avalia o equilíbrio entre precisão e recall, foi de 0.00%.

Além disso, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 18.54%, indicando um desempenho inferior ao de um classificador aleatório.

Comitê de Redes Neurais Artificiais (Ensemble of Artificial Neural Networks)

O Comitê de Redes Neurais Artificiais é uma abordagem avançada de aprendizado de máquina que combina várias redes neurais para melhorar a precisão e a robustez das previsões. Em vez de depender de um único modelo, essa técnica utiliza múltiplas redes neurais, treinadas de maneira independente, cujas previsões são então agregadas para produzir um resultado final mais confiável. A ideia é que cada rede neural possa capturar diferentes aspectos dos dados e erros individuais dos modelos possam ser compensados pelo desempenho de outros. As combinações podem ser feitas de várias formas, incluindo médias ponderadas, votação majoritária ou métodos mais complexos como stacking, onde um modelo adicional é usado para combinar as previsões das redes neurais. Essa abordagem pode levar a melhorias significativas na performance, especialmente em problemas complexos e com dados ruidosos, mas também pode aumentar a complexidade computacional e a dificuldade de interpretação do modelo.

Hiperparâmetros e Treinamento

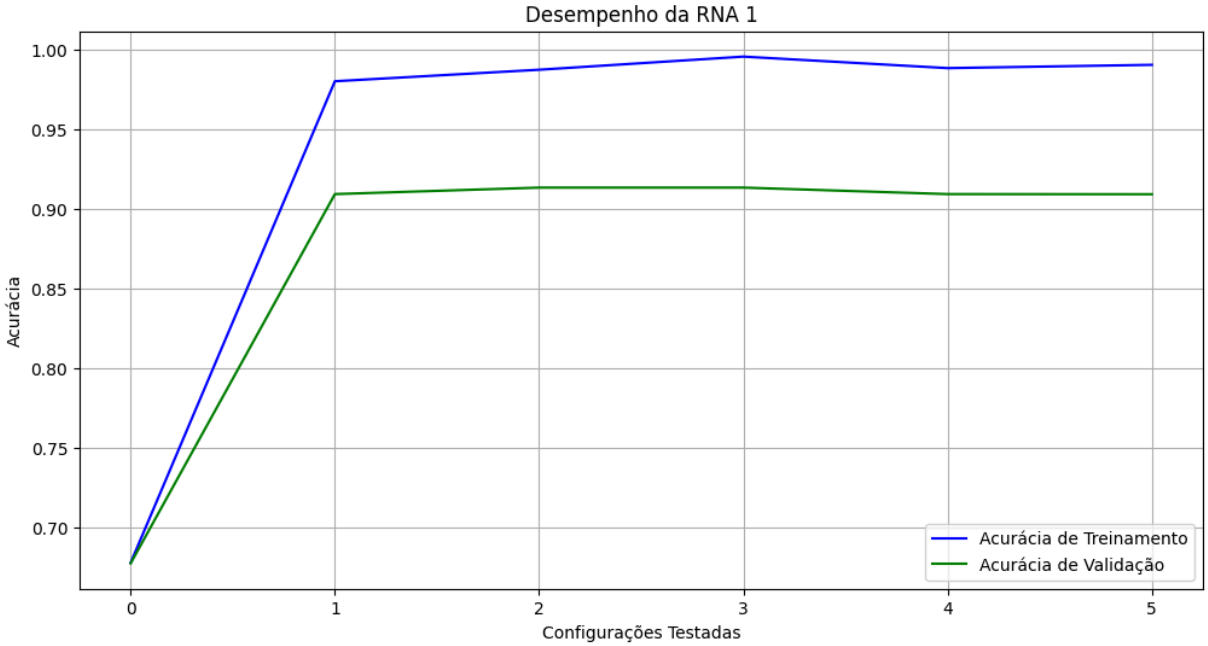
In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

# Definindo múltiplas redes neurais para o comitê
nn1 = MLPClassifier(random_state=42)
nn2 = MLPClassifier(random_state=42)
nn3 = MLPClassifier(random_state=42)

# Grid Search para encontrar os melhores hiperparâmetros
param_grid = {
    'hidden_layer_sizes': [(100,), (150,), (200,)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam']
}

grid_nn1 = GridSearchCV(nn1, param_grid, cv=5, scoring='accuracy', return_tr
grid_nn2 = GridSearchCV(nn2, param_grid, cv=5, scoring='accuracy', return_tr
grid_nn3 = GridSearchCV(nn3, param_grid, cv=5, scoring='accuracy', return_tr

# Treinamento dos modelos com os melhores hiperparâmetros
grid_nn1.fit(X_train, y_train)
grid_nn2.fit(X_train, y_train)
grid_nn3.fit(X_train, y_train)

best_nn1 = grid_nn1.best_estimator_
best_nn2 = grid_nn2.best_estimator_
best_nn3 = grid_nn3.best_estimator_

# Comitê de Redes Neurais Artificiais utilizando VotingClassifier
ensemble_nn = VotingClassifier(estimators=[
    ('nn1', best_nn1),
    ('nn2', best_nn2),
    ('nn3', best_nn3)
], voting='soft')
```

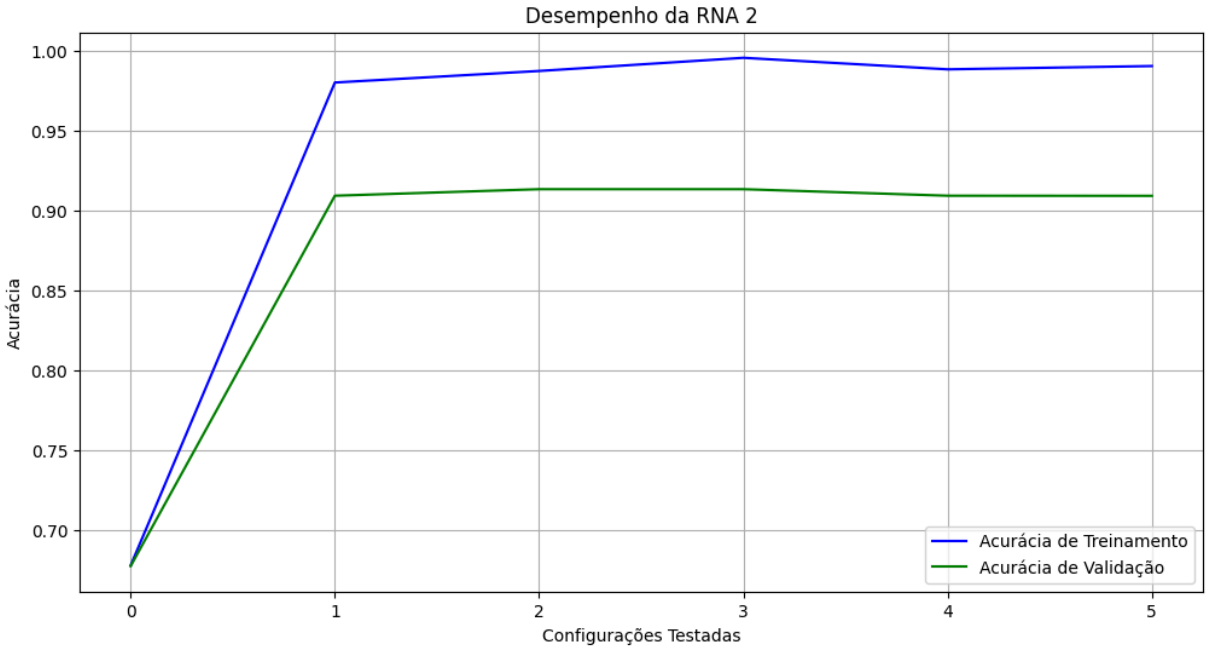
```
# Treinamento do Comitê de Redes Neurais Artificiais
ensemble_nn.fit(X_train, y_train)

# Predições e avaliação
y_pred_train_nn = ensemble_nn.predict(X_train)
y_pred_valid_nn = ensemble_nn.predict(X_valid)
y_pred_test_nn = ensemble_nn.predict(X_test)
```

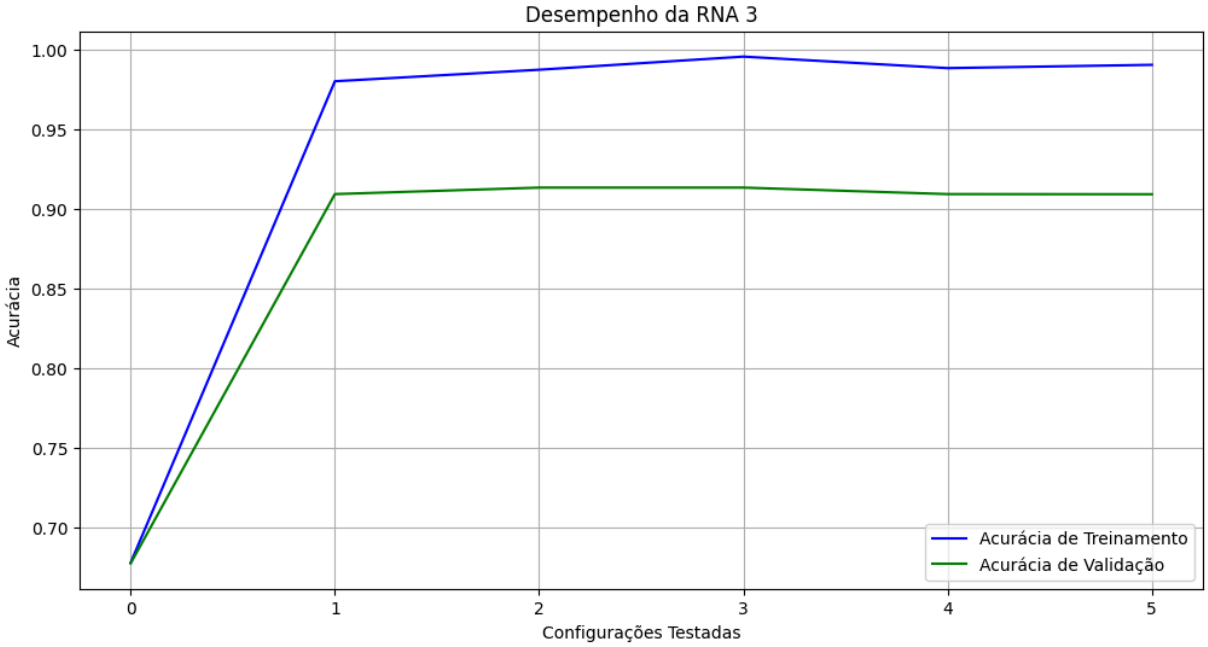
Apresentando os resultados

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

# Obter resultados do GridSearch para cada rede neural
results_nn1 = grid_nn1.cv_results_
results_nn2 = grid_nn2.cv_results_
results_nn3 = grid_nn3.cv_results_

# Função para plotar resultados de GridSearch
def plot_grid_search_results(results, title):
    # Pegando as médias dos resultados de cross-validation
    mean_train_score = results['mean_train_score']
    mean_test_score = results['mean_test_score']

    # Número de configurações testadas
    n_configs = len(mean_train_score)

    plt.figure(figsize=(12, 6))
    x = np.arange(n_configs)
    plt.plot(x, mean_train_score, label='Acurácia de Treinamento', color='bl
    plt.plot(x, mean_test_score, label='Acurácia de Validação', color='green

    plt.title(title)
    plt.xlabel('Configurações Testadas')
    plt.ylabel('Acurácia')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

# Plotando os resultados para cada rede neural
plot_grid_search_results(results_nn1, 'Desempenho da RNA 1')
plot_grid_search_results(results_nn2, 'Desempenho da RNA 2')
plot_grid_search_results(results_nn3, 'Desempenho da RNA 3')
```


Desempenho da RNA 1
Acurácia
Configurações Testadas
Acurácia de Treinamento
Acurácia de Validação


Desempenho da RNA 2
Acurácia
Configurações Testadas
Acurácia de Treinamento
Acurácia de Validação



In [ ]:
```python
# Salvando as métricas de desempenho em um dicionário
current_results_nn = {
    "Classificador": "Comitê RN",
    "Acurácia de Treinamento": accuracy_score(y_train, y_pred_train_nn),
    "Acurácia de Validação": accuracy_score(y_valid, y_pred_valid_nn),
    "Acurácia de Teste": accuracy_score(y_test, y_pred_test_nn),
    "Precision": precision_score(y_test, y_pred_test_nn, pos_label=1),
    "Recall": recall_score(y_test, y_pred_test_nn, pos_label=1),
    "F1": f1_score(y_test, y_pred_test_nn, pos_label=1),
    "AUC-ROC": roc_auc_score(y_test, ensemble_nn.predict_proba(X_test)[:, 1]
}

# Adicionar os resultados do RNA à lista de resultados
results.append(current_results_nn)

# Imprimindo os resultados do comitê de redes neurais
print("Desempenho do Comitê de Redes Neurais Artificiais:")
for metric, value in current_results_nn.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")
```

```
Desempenho do Comitê de Redes Neurais Artificiais:
Classificador: Comitê RN
Acurácia de Treinamento: 98.35%
Acurácia de Validação: 91.11%
Acurácia de Teste: 91.30%
Precision: 100.00%
Recall: 20.00%
F1: 33.33%
AUC-ROC: 90.24%
```

pós ajustar o Comitê de Redes Neurais Artificiais e explorar suas configurações, observamos que o modelo atingiu uma acurácia de 98.35% nos dados de treinamento, 91.11% nos dados

de validação e 91.30% nos dados de teste. A precisão foi de 100.00%, indicando que todas as instâncias positivas foram corretamente classificadas como positivas. No entanto, o recall foi de 20.00%, sugerindo que o modelo conseguiu identificar apenas 20% das instâncias positivas reais. O F1-score foi de 33.33%, refletindo o equilíbrio entre a precisão e o recall.

Adicionalmente, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 90.24%, indicando um desempenho superior ao de um classificador aleatório.

Comitê Heterogêneo (Heterogeneous Ensemble)

O Comitê Heterogêneo é uma técnica de aprendizado de máquina que combina diferentes tipos de modelos para aproveitar suas respectivas forças e superar suas fraquezas individuais. Em vez de usar múltiplas instâncias do mesmo tipo de modelo, o comitê heterogêneo agrega previsões de variados algoritmos, como árvores de decisão, máquinas de vetores de suporte e redes neurais, entre outros. Cada modelo contribui com uma perspectiva única sobre os dados, e a combinação das suas previsões pode resultar em um desempenho geral mais robusto e preciso. Os métodos de agregação podem incluir votação majoritária, média ponderada ou métodos de meta-aprendizagem como stacking, onde um modelo adicional aprende a melhor forma de combinar as previsões dos modelos base. Embora a utilização de um comitê heterogêneo possa melhorar significativamente a performance e a robustez, ela também pode aumentar a complexidade do sistema e os requisitos computacionais.

Hiperparâmetros e Treinamento

In [ ]:
```
!pip install --upgrade pip
!pip install auto-sklearn
```

In [ ]:
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
import sklearn.utils
import autosklearn.classification

# Definir o modelo de AutoML
automl = autosklearn.classification.AutoSklearnClassifier(
    time_left_for_this_task=300,  # Tempo total de execução em segundos
    per_run_time_limit=30,        # Tempo limite por modelo em segundos
    n_jobs=-1,                    # Usar todos os núcleos disponíveis
)

# Treinar o modelo
automl.fit(X_train, y_train)

best_model = automl.get_models_with_weights()[0][1]
```

```
y_pred_test = best_model.predict(X_test)
y_pred_train = best_model.predict(X_train)
y_pred_valid = best_model.predict(X_valid)

y_pred_test_proba = best_model.predict_proba(X_test)[:, 1] if hasattr(best_m
```

Apresentando os resultados

In [ ]:
```
current_results = {
    "Classificador": "Comitê Heterogêneo",
    "Acurácia de Treinamento": accuracy_score(y_train, y_pred_train),
    "Acurácia de Validação": accuracy_score(y_valid, y_pred_valid),
    "Acurácia de Teste": accuracy_score(y_test, y_pred_test),
    "Precision": precision_score(y_test, y_pred_test, pos_label=1),
    "Recall": recall_score(y_test, y_pred_test, pos_label=1),
    "F1": f1_score(y_test, y_pred_test, pos_label=1),
    "AUC-ROC": roc_auc_score(y_test, y_pred_test_proba) if y_pred_test_proba
}

# Imprimindo os resultados do comitê heterogêneo
print("Desempenho do Comitê Heterogêneo:")
for metric, value in current_results.items():
    if isinstance(value, float):
        print(f"{metric}: {value * 100:.2f}%")
    else:
        print(f"{metric}: {value}")

# Adicionar os resultados do Comitê à lista de resultados
results.append(current_results)
```

```
Desempenho do Comitê Heterogêneo:
Classificador: Comitê Heterogêneo
Acurácia de Treinamento: 97.11%
Acurácia de Validação: 80.00%
Acurácia de Teste: 93.48%
Precision: 75.00%
Recall: 60.00%
F1: 66.67%
AUC-ROC: 87.80%
```

Após ajustar o modelo Comitê Heterogêneo e testar diferentes configurações, identificamos que ele apresentou um desempenho notável em várias métricas.

O modelo alcançou uma acurácia de 97.11% nos dados de treinamento, 80.00% nos dados de validação e 93.48% nos dados de teste. Esses resultados indicam que o modelo se ajustou bem aos dados de treinamento e mostrou uma boa capacidade de generalização nos dados de teste. A precisão foi de 75.00%, o que significa que o modelo classificou corretamente 75 de cada 100 clientes que foram identificados como potenciais para depósitos a prazo. O recall foi de 60.00%, refletindo a capacidade do modelo em identificar corretamente 60% dos clientes que realmente fazem depósitos a prazo. O F1-score, que é a

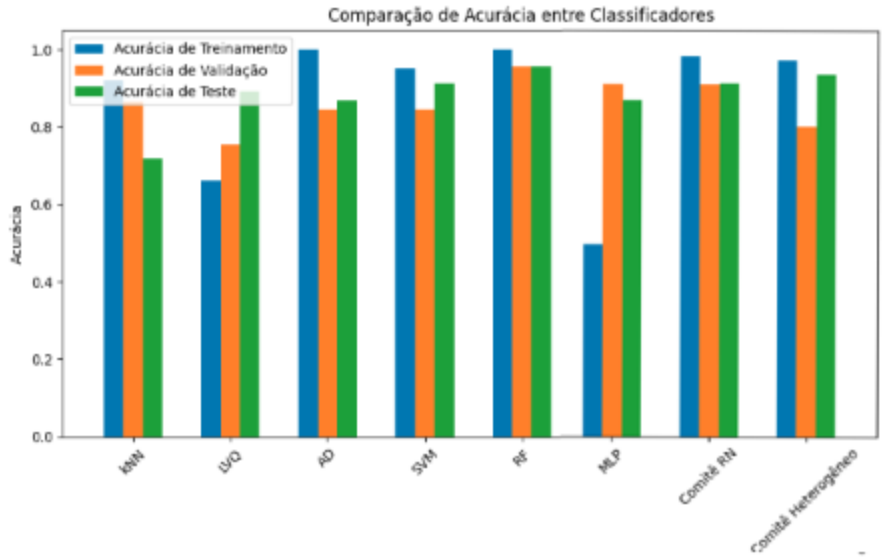
média harmônica entre precisão e recall, foi de 66.67%, oferecendo um equilíbrio entre esses dois aspectos.

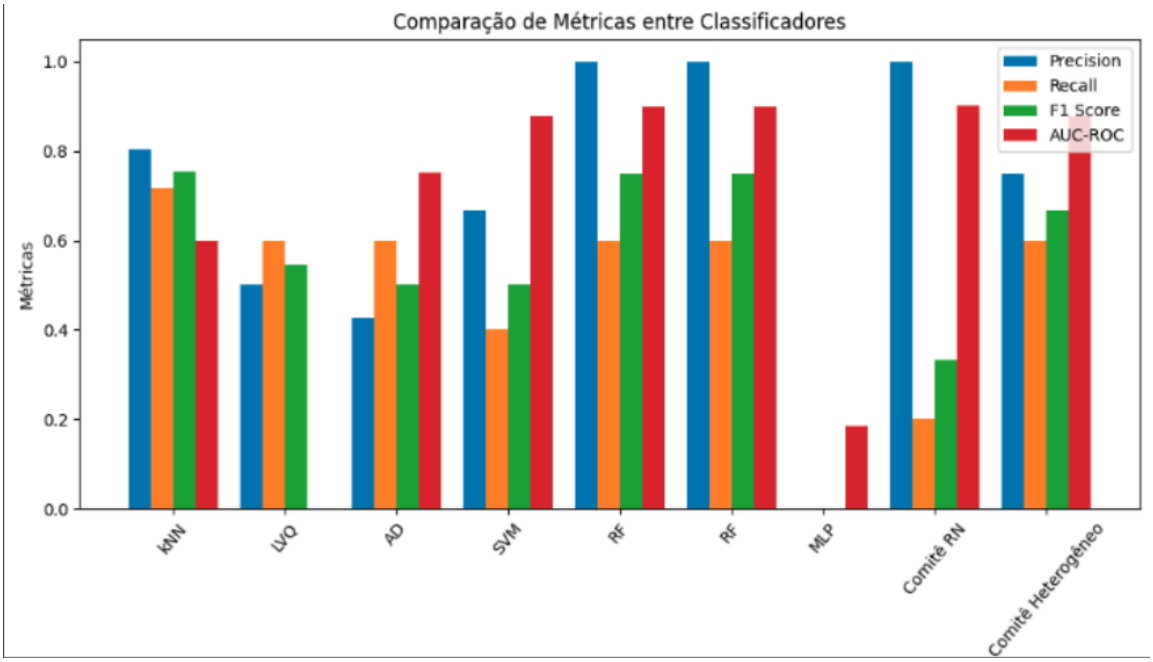
Além disso, a análise da área sob a curva ROC (AUC-ROC) revelou um valor de 87.80%, indicando que o modelo possui uma alta capacidade de discriminação entre as classes positiva e negativa, superando o desempenho de um classificador aleatório.

Comparando os indicadores dos classificadores (resultados)

In [ ]:
```
# Definindo os classificadores e as métricas
classificadores = [result["Classificador"] for result in results]
acuracias_treinamento = [result["Acurácia de Treinamento"] for result in res
acuracias_validacao = [result["Acurácia de Validação"] for result in results
acuracias_teste = [result["Acurácia de Teste"] for result in results]
precisions = [result["Precision"] for result in results]
recalls = [result["Recall"] for result in results]
f1_scores = [result["F1"] for result in results]
auc_rocs = [result["AUC-ROC"] for result in results]

# Configurando a posição dos gráficos
x = np.arange(len(classificadores))

# Largura das barras
width = 0.2

# Plotando as acurácias
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x - width, acuracias_treinamento, width, label='Acurácia de Treinamen
ax.bar(x, acuracias_validacao, width, label='Acurácia de Validação')
ax.bar(x + width, acuracias_teste, width, label='Acurácia de Teste')

# Adicionando labels e título
ax.set_xlabel('Classificadores')
ax.set_ylabel('Acurácia')
ax.set_title('Comparação de Acurácia entre Classificadores')
ax.set_xticks(x)
ax.set_xticklabels(classificadores)
ax.legend()

# Exibindo o gráfico
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Plotando Precision, Recall, F1 e AUC-ROC
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(x - width*1.5, precisions, width, label='Precision')
ax.bar(x - width/2, recalls, width, label='Recall')
ax.bar(x + width/2, f1_scores, width, label='F1 Score')
ax.bar(x + width*1.5, auc_rocs, width, label='AUC-ROC')
```

```
# Adicionando labels e título
ax.set_xlabel('Classificadores')
ax.set_ylabel('Métricas')
ax.set_title('Comparação de Métricas entre Classificadores')
ax.set_xticks(x)
ax.set_xticklabels(classificadores)
ax.legend()

# Exibindo o gráfico
plt.xticks(rotation=50)
plt.tight_layout()
plt.show()
```





Seleção do Modelo Final

A escolha do modelo final não será determinada apenas pela acurácia nos dados de treinamento e teste, mas também por uma análise abrangente de várias métricas de desempenho. A **precisão** é um critério essencial, pois avalia a capacidade do modelo de identificar corretamente os clientes que efetivamente realizam depósitos a prazo. Este aspecto é fundamental, dado que o objetivo principal do projeto é identificar esses clientes com alta precisão.

Além disso, o **recall** é uma métrica importante, pois mede a proporção de clientes que realmente fazem depósitos a prazo e que o modelo consegue identificar corretamente. O recall reflete a habilidade do modelo em captar todos os verdadeiros positivos, ou seja, os clientes que estão dispostos a realizar um depósito a prazo.

O **F1-score** será utilizado para equilibrar a precisão e o recall, oferecendo uma visão mais completa do desempenho do modelo. Esta métrica é particularmente útil quando se busca um equilíbrio entre identificar corretamente os clientes que fazem depósitos a prazo e minimizar a quantidade de falsos positivos e falsos negativos.

Finalmente, a métrica **AUC-ROC** será empregada para avaliar a capacidade do modelo de distinguir entre as classes positiva e negativa, refletindo a probabilidade de o modelo identificar corretamente a classe positiva ao longo de diferentes limiares de decisão.

Com base na análise dessas métricas, o modelo **Random Forest (RF)** com profundidade máxima de 124 e 206 estimadores se destacou como o melhor para o nosso objetivo. Este modelo apresentou uma precisão de 100%, indicando que não classificou nenhum cliente que não faria um depósito a prazo como propenso a fazê-lo. No entanto, devido ao desbalanceamento entre as classes, o recall foi de 60%. Apesar disso, o F1-score de 75% e o AUC-ROC de 90% evidenciam que o Random Forest oferece o melhor equilíbrio entre identificação precisa e abrangente dos clientes interessados em depósitos a prazo, alinhando-se mais efetivamente com os objetivos do projeto.

Reflexões Críticas

Durante o desenvolvimento e avaliação do modelo, enfrentamos alguns desafios significativos que impactaram sua eficiência e desempenho. Um dos principais obstáculos foi a limitação de recursos computacionais, que impediu a utilização completa da base de dados. Apesar dessa restrição, conseguimos construir um modelo funcional que pode auxiliar na tomada de decisões.

Além disso, o desequilíbrio entre as classes de depósito a prazo e não depósito a prazo foi um fator crítico a ser considerado. Esse desbalanceamento afetou as métricas de avaliação, como a precisão e o recall, tornando crucial a análise detalhada de cada métrica para entender o desempenho real do modelo. A métrica de precisão, que avalia a capacidade do modelo de identificar corretamente os clientes que fazem depósitos a prazo, é

especialmente importante para o objetivo do projeto, que é direcionar campanhas de telemarketing mais eficazes.

O modelo deve ser capaz de ajudar a equipe de marketing do banco a identificar clientes com maior propensão a aceitar depósitos a prazo, melhorando a eficiência das campanhas de telemarketing e aumentando a taxa de conversão. O recall é uma métrica crucial neste contexto, pois indica a proporção de clientes que realmente aceitariam fazer um depósito a prazo, mas que foram classificados incorretamente como não interessados.

O F1-score oferece um equilíbrio entre precisão e recall, fornecendo uma visão mais completa do desempenho do modelo. Além disso, a AUC-ROC revela a capacidade do modelo de distinguir entre as classes positivas e negativas, ajudando a avaliar sua eficácia geral.

Em suma, apesar das limitações e desafios enfrentados, o modelo desenvolvido representa um avanço importante para apoiar a equipe de marketing do banco na otimização das campanhas de telemarketing e na captação de novos clientes para depósitos a prazo. Ajustes futuros e a implementação de técnicas para lidar com o desequilíbrio de classes podem aprimorar ainda mais a eficácia do modelo.

Conclusão

Neste projeto, o objetivo principal foi desenvolver e avaliar um modelo de machine learning para auxiliar a equipe de marketing do banco na realização de campanhas de telemarketing mais eficazes, com foco na captação de novos clientes para depósitos a prazo. Através da análise e comparação de vários modelos, conseguimos identificar o que melhor atende a essa necessidade específica.

A etapa de preparação dos dados foi crucial para o sucesso do projeto. Um tratamento adequado dos dados, incluindo a limpeza, normalização e manipulação das variáveis, estabeleceu uma base sólida para o ajuste e a avaliação do modelo. A preparação cuidadosa dos dados garantiu que o modelo fosse treinado com informações relevantes e de alta qualidade, o que contribuiu significativamente para a performance geral dos modelos avaliados.

O modelo de Random Forest (RF) com profundidade máxima de 124 e 206 estimadores apresentou o melhor desempenho geral, com acurácia elevada tanto nos dados de treinamento quanto nos de validação e teste. No entanto, a análise das métricas de desempenho revelou que, apesar da alta acurácia, o modelo precisa ser ajustado para melhorar o recall, especialmente devido ao desequilíbrio entre as classes. A precisão, recall, e o F1-score forneceram insights valiosos sobre a capacidade do modelo de identificar corretamente os clientes interessados em depósitos a prazo, enquanto a AUC-ROC ajudou a avaliar a qualidade geral das previsões.

A avaliação crítica do modelo destacou algumas limitações, como a falta de recursos computacionais que restringiu a capacidade de trabalhar com toda a base de dados e o desequilíbrio das classes, que afetou as métricas de desempenho. Apesar desses desafios, o modelo desenvolvido oferece uma base sólida para aprimorar as campanhas de telemarketing e direcionar esforços de marketing de forma mais eficaz.

Para futuros aprimoramentos, recomendamos explorar técnicas adicionais para lidar com o desequilíbrio das classes, como reamostragem ou ajuste de pesos das classes. Também seria benéfico expandir a base de dados e considerar a implementação de modelos mais complexos para potencialmente aumentar a precisão e a capacidade de generalização do modelo.

Em resumo, este projeto estabeleceu uma base para melhorar a eficácia das campanhas de telemarketing e otimizar a captação de clientes para depósitos a prazo, oferecendo insights importantes para a equipe de marketing do banco e indicando direções promissoras para futuros desenvolvimentos.