Projeto Integrador Parte E

Alunas:

- **Gabriella Braz**
- **Giovana Ribeiro**

1. Use o mesmo conjunto de dados já escolhido anteriormente ou escolha um novo conjunto de dados.

```
data_path = Path("../../../data/tb_1.csv")
df = pd.read_csv(data_path)

print("Dimensão do dataset:", df.shape)
df.head()
```

✓ 0.0s Python

```
Dimensão do dataset: (1000, 8)
```

	gender	race_ethnicity	parental_level_of_education	lunch	test_preparation_course	math_score	reading_score	writing_score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

2. Implemente os algoritmos Naive Bayes, KNN e árvore de decisão para classificação.

```
# 1. NAIVE BAYES
print("Treinando Naive Bayes...")
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
nb_pred = nb_model.predict(X_test_scaled)
nb_prob = nb_model.predict_proba(X_test_scaled)
print("Naive Bayes treinado!")
```

✓ 0.0s Python

```
Treinando Naive Bayes...
Naive Bayes treinado!
```

```
# 2. KNN
print("Treinando KNN...")
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)
knn_prob = knn_model.predict_proba(X_test_scaled)
print("KNN treinado!")
```

✓ 0.1s Python

```
Treinando KNN...
KNN treinado!
```

```
# 3. ÁRVORE DE DECISÃO
print("Treinando Árvore de Decisão...")
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train)
dt_pred = dt_model.predict(X_test_scaled)
dt_prob = dt_model.predict_proba(X_test_scaled)
print("Árvore de Decisão treinada!")
```

✓ 0.0s Python

```
Treinando Árvore de Decisão...
Árvore de Decisão treinada!
```

3. Analise a matriz de confusão, acurácia, precisão, recall, f1_score e área sob a curva roc para todos os algoritmos.

```python
from sklearn.metrics import (
    confusion_matrix, accuracy_score, precision_score,
    recall_score, f1_score, roc_auc_score, roc_curve
)

def evaluate_model(y_test, y_pred, y_prob, model_name):
    print(f"\n===== {model_name} =====")
    cm = confusion_matrix(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    rec = recall_score(y_test, y_pred, average='weighted', zero_division=0)
    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)

    print("Matriz de Confusão:\n", cm)
    print(f"Acurácia: {acc:.4f}")
    print(f"Precisão: {prec:.4f}")
    print(f"Recall: {rec:.4f}")
    print(f"F1-Score: {f1:.4f}")

    # Verifica se é binário para calcular ROC
    if len(set(y_test)) == 2 and y_prob is not None:
        try:
            roc_auc = roc_auc_score(y_test, y_prob[:, 1])
            print(f"Área sob a curva ROC: {roc_auc:.4f}")
        except Exception as e:
            print("Não foi possível calcular a curva ROC:", e)
    else:
        print("Curva ROC não aplicável (problema multiclasse).")
```

✓ 0.0s Python

```python
def calculate_all_metrics(y_test, y_pred, y_prob, class_names):
    """Calcula todas as métricas para um modelo"""
    metrics = {}

    # Métricas básicas
    metrics['confusion_matrix'] = confusion_matrix(y_test, y_pred)
    metrics['accuracy'] = accuracy_score(y_test, y_pred)
    metrics['precision_macro'] = precision_score(y_test, y_pred, average='macro', zero_division=0)
    metrics['recall_macro'] = recall_score(y_test, y_pred, average='macro', zero_division=0)
    metrics['f1_macro'] = f1_score(y_test, y_pred, average='macro', zero_division=0)

    # ROC AUC
    try:
        if len(class_names) == 2:  # Binário
            metrics['roc_auc'] = roc_auc_score(y_test, y_prob[:, 1])
        else:  # Multiclasse
            # Binarizar para multiclasse
            y_test_bin = label_binarize(y_test, classes=range(len(class_names)))
            metrics['roc_auc'] = roc_auc_score(y_test_bin, y_prob, multi_class='ovr', average='macro')
    except:
        metrics['roc_auc'] = "N/A"

    return metrics

print("Função de cálculo de métricas definida!")
```

✓ 0.0s Python

```
Função de cálculo de métricas definida!
```

```python
# Calcular métricas para todos os modelos
results = {
    "Naive Bayes": calculate_all_metrics(y_test, nb_pred, nb_prob, class_names),
    "KNN": calculate_all_metrics(y_test, knn_pred, knn_prob, class_names),
    "Árvore de Decisão": calculate_all_metrics(y_test, dt_pred, dt_prob, class_names)
}

# Preparar dados para curvas ROC
def prepare_roc_data(y_test, y_prob, class_names):
    """Prepara dados para curva ROC"""
    if len(class_names) == 2:  # Problema binário
        fpr, tpr, _ = roc_curve(y_test, y_prob[:, 1])
        return fpr, tpr
    else:  # Problema multiclasse - usar macro average
        try:
            y_test_bin = label_binarize(y_test, classes=range(len(class_names)))
            fpr, tpr, _ = roc_curve(y_test_bin.ravel(), y_prob.ravel())
            return fpr, tpr
        except:
            return None, None

y_probas = {
    "Naive Bayes": prepare_roc_data(y_test, nb_prob, class_names),
    "KNN": prepare_roc_data(y_test, knn_prob, class_names),
    "Árvore de Decisão": prepare_roc_data(y_test, dt_prob, class_names)
}

print("Métricas calculadas e dados ROC preparados!")
```

✓ 0.4s Python

```python
for name, metrics in results.items():
    print(f"\n== {name} ==")
    print("Matriz de confusão:\n", metrics["confusion_matrix"])
    print(f"Acurácia: {metrics['accuracy']:.4f}")
    print(f"Precisão (macro): {metrics['precision_macro']:.4f}")
    print(f"Recall (macro): {metrics['recall_macro']:.4f}")
    print(f"F1-score (macro): {metrics['f1_macro']:.4f}")
    print(f"ROC AUC: {metrics['roc_auc']}")
```

✓ 0.0s Python

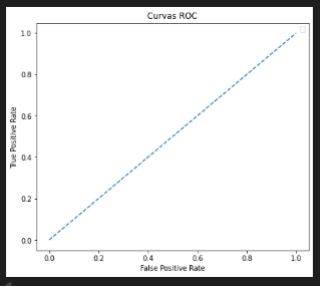
```
== Naive Bayes ==
Matriz de confusão:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]]
Acurácia: 0.0200
Precisão (macro): 0.0067
Recall (macro): 0.0173
F1-score (macro): 0.0094
ROC AUC: nan

== KNN ==
Matriz de confusão:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 1]]
Acurácia: 0.0320
Precisão (macro): 0.0266
Recall (macro): 0.0268
F1-score (macro): 0.0228
ROC AUC: nan
```

```
== Árvore de Decisão ==
Matriz de confusão:
 [[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 1 0]]
Acurácia: 0.0640
Precisão (macro): 0.0376
Recall (macro): 0.0484
F1-score (macro): 0.0358
ROC AUC: nan
```

```
plt.figure(figsize=(7,6))
for name, (fpr, tpr) in y_probas.items():
    if fpr is not None:
        plt.plot(fpr, tpr, label=f"{name} (AUC={results[name]['roc_auc']:.3f})")
plt.plot([0,1],[0,1],'--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Curvas ROC")
plt.legend()
plt.show()
```

0.9s Python

```
C:\Users\Giovana\AppData\Local\Temp\ipykernel_47836\2853814614.py:9: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend
  plt.legend()
```



4. Entregue os códigos com as análises usando as métricas acima.

5. Entregue um relatório com a discussão sobre os resultados obtidos, comparando o desempenho dos algoritmos e interpretando os impactos das métricas analisadas na tomada de decisão.

   Os resultados obtidos com os modelos testados: Naive Bayes, KNN e Árvore de Decisão, evidenciam as dificuldades em classificar corretamente os dados disponíveis. O modelo Naive Bayes apresentou desempenho bastante limitado, com acurácia de apenas 2%, além de valores baixos de precisão (0,67%), recall (1,73%) e F1-score (0,94%). Esses indicadores mostram que o modelo não conseguiu capturar relações significativas entre as variáveis, provavelmente devido à forte correlação entre atributos como notas de matemática, leitura e escrita, o que contraria a suposição de independência condicional do Naive Bayes.

   O modelo KNN obteve métricas ligeiramente superiores às do Naive Bayes, alcançando acurácia de 3,2%, precisão de 2,66%, recall de 2,68% e F1-score de 2,28%. Embora tenha demonstrado uma pequena melhora, os valores ainda são baixos, sugerindo que os vizinhos mais próximos não conseguiram capturar padrões claros no espaço de atributos. Isso pode estar relacionado ao grande número de classes e ao desbalanceamento dos dados, já que algumas classes apresentavam apenas uma amostra, dificultando a generalização.

Já a Árvore de Decisão foi o algoritmo com o melhor desempenho relativo, atingindo acurácia de 6,4%, precisão de 3,76%, recall de 4,84% e F1-score de 3,58%. Esse resultado indica que o modelo conseguiu explorar de forma mais adequada as relações entre as variáveis categóricas e numéricas, ainda que os valores permaneçam muito baixos para uma aplicação prática.

Em relação à curva ROC, não foi possível extrair informações relevantes, dado que se trata de um problema multiclasse e a AUC não pôde ser calculada de forma consistente. Assim, as métricas de acurácia, precisão, recall e F1-score são mais adequadas para a análise deste cenário.

De modo geral, os resultados indicam que nenhum dos três modelos apresentou desempenho satisfatório para a tarefa de classificação proposta. Ainda assim, a Árvore de Decisão demonstrou-se mais promissora e poderia servir como base para algoritmos mais robustos, como Random Forest ou Gradient Boosting. Além disso, estratégias como o balanceamento das classes, a redução do número de categorias ou a redefinição do problema em menos classes podem contribuir para melhorar os resultados.