Relatório

Sobre o dataset

O conjunto de dados "Wine Quality" disponível no repositório UCI Machine Learning contém informações sobre vinhos portugueses de duas variantes: vinho tinto e vinho branco. No entanto, para este projeto, utilizaremos exclusivamente os dados relacionados aos vinhos tintos.

Descrição do Conjunto de Dados

Cada registro no conjunto de dados representa uma amostra de vinho e inclui as seguintes características:

1. **fixed acidity**: Acidez fixa (mais estável e não volátil).
2. **volatile acidity**: Acidez volátil (responsável pelo aroma de vinagre).
3. **citric acid**: Ácido cítrico (encontrado em pequenas quantidades).
4. **residual sugar**: Açúcar residual (quantidade de açúcar que resta após a fermentação).
5. **chlorides**: Cloretos (quantidade de sal).
6. **free sulfur dioxide**: Dióxido de enxofre livre.
7. **total sulfur dioxide**: Dióxido de enxofre total.
8. **density**: Densidade do vinho.
9. **pH**: Medida de acidez/alcalinidade.
10. **sulphates**: Sulfatos (aditivos do vinho).
11. **alcohol**: Percentual de álcool.
12. **quality**: Qualidade do vinho (avaliação sensorial, pontuação entre 0 e 10).

Além disso, procuramos por valores nulos no conjunto de dados, porém, felizmente, nenhum valor nulo foi encontrado.

Contagem da coluna quality

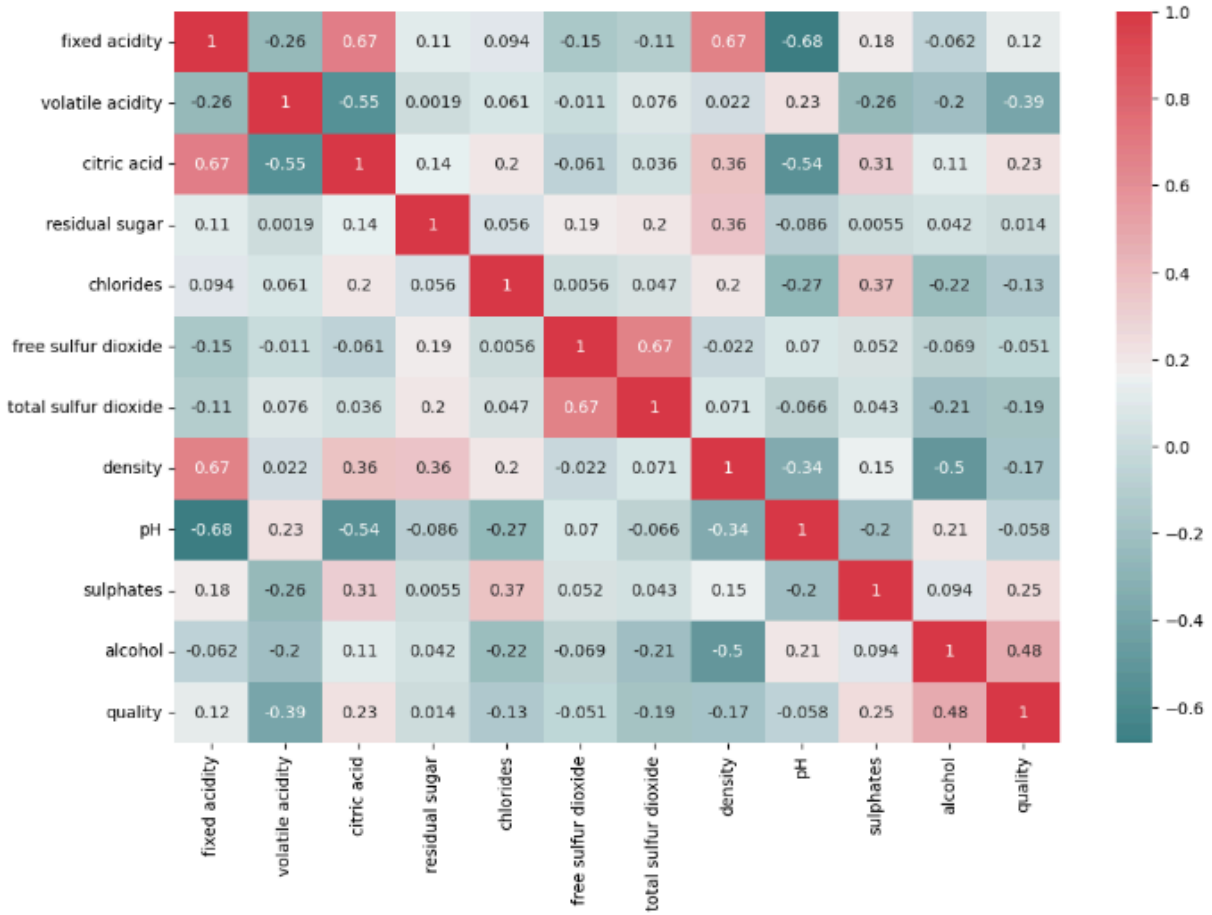
Na análise do dataset, também procuramos saber da quantidade de ocorrência de cada valor da coluna quality. Segue o que encontramos:

- quality 3: 10 ocorrências
- quality 4: 53 ocorrências
- quality 5: 681 ocorrências
- quality 6: 638 ocorrências
- quality 7: 199 ocorrências
- quality 8: 18 ocorrências

Com isso, é possível afirmar que o modelo trará melhores resultados para classificar vinhos de qualidade 5 ou 6, já que possuem uma maior amostragem

Análise exploratória

Para terminar a análise do dataset, realizamos uma análise exploratória e checamos a correlação entre colunas:



A partir dela, podemos analisar que o teor alcoólico possui alta correlação com a qualidade do vinho. E, além disso, podemos ver que o açúcar residual, pelo contrário, possui uma correlação próxima a 0 quando comparada com a qualidade. Porém, mesmo assim, optamos por não removermos essa coluna, já que modelos como árvores de decisão podem capturar interações complexas entre features, mesmo que a princípio elas aparentam ser irrelevantes.

Classificadores usados

Árvores de decisão

Desempenho inicial

Quando utilizamos a árvore de decisão como classificador, primeiro realizamos o treinamento do modelo utilizando os hiperparâmetros base da ferramenta do sklearn. Esse foi o resultado que obtivemos:

	precision	recall	f1-score	support
3	0.0000	0.0000	0.0000	3
4	0.0526	0.0625	0.0571	16
5	0.7143	0.6618	0.6870	204
6	0.6202	0.6719	0.6450	192
7	0.5577	0.4833	0.5179	60
8	0.2222	0.4000	0.2857	5
accuracy			0.6167	480
macro avg	0.3612	0.3799	0.3655	480
weighted avg	0.6254	0.6167	0.6196	480

Grid Search

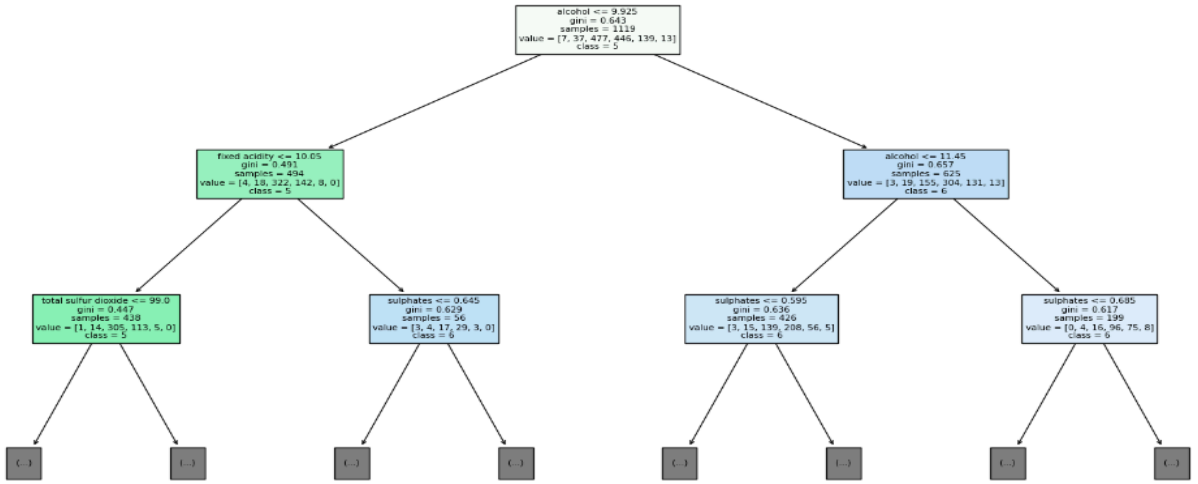
A partir disso, procuramos melhorar o modelo utilizando o Grid Search CV, que é uma técnica de otimização de hiperparâmetros amplamente utilizada em aprendizado de máquina. "Grid" refere-se ao fato de que ele examina todas as combinações possíveis de hiperparâmetros especificados em uma grade ou grade de valores, e "CV" refere-se à validação cruzada que é usada para avaliar o desempenho do modelo para cada combinação de hiperparâmetros. No nosso caso, nosso Grid Search irá procurar melhorar o f1-score macro. Segue a lista de hiperparâmetros utilizados no grid:

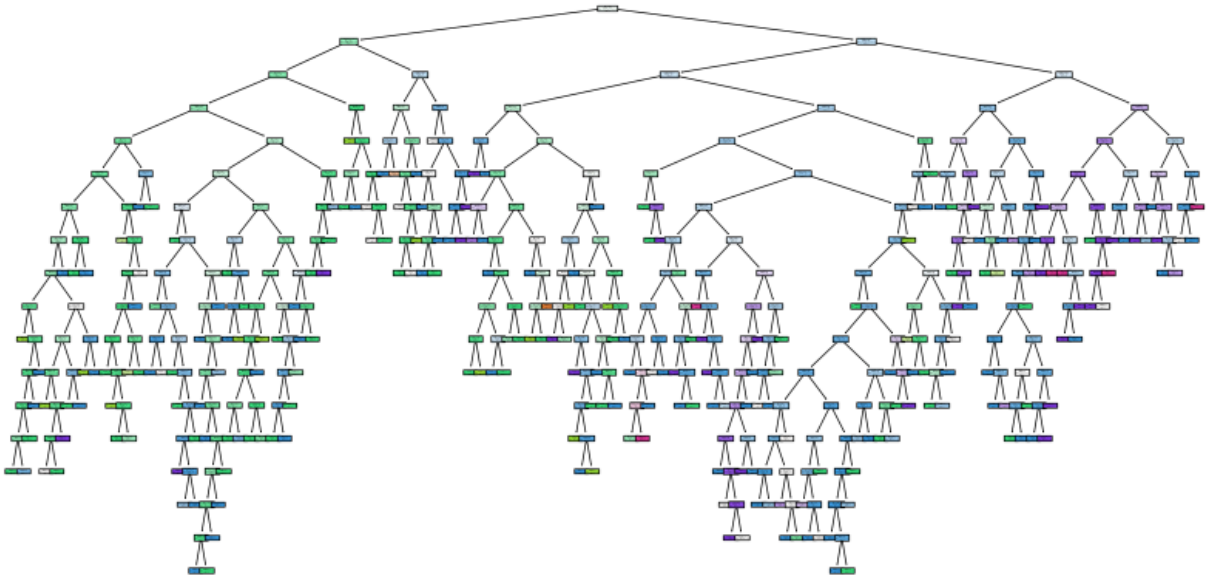
```
[62] # Definindo o grid de parâmetros
     param_grid = {
         'max_depth': [None, 10, 20, 30, 40, 50],
         'min_samples_split': [2,5, 10, 20],
         'min_samples_leaf': [1, 2, 5, 10,20],
         'criterion': ['gini', 'entropy']
     }

     # Inicializando o GridSearchCV
     grid_search = GridSearchCV(DT_clf, param_grid, cv=5, scoring='f1_macro', n_jobs=-1)
```

Desempenho final

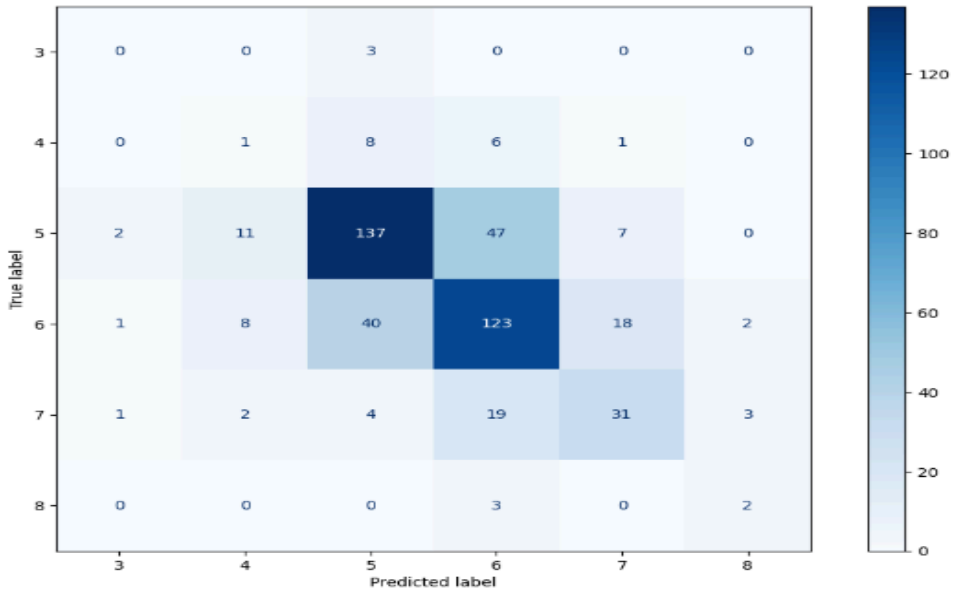
Após isso, obtivemos um resultado que possui uma melhora geral no F1-score, porém não grande o suficiente para ser significativa. Então, no geral, o resultado se

manteve parecido. Segue as imagens da árvore de decisão final e seu resultado seguido pela matriz de confusão:





	precision	recall	f1-score	support
3	0.0000	0.0000	0.0000	3
4	0.0455	0.0625	0.0526	16
5	0.7135	0.6716	0.6919	204
6	0.6212	0.6406	0.6308	192
7	0.5439	0.5167	0.5299	60
8	0.2857	0.4000	0.3333	5
accuracy			0.6125	480
macro avg	0.3683	0.3819	0.3731	480
weighted avg	0.6242	0.6125	0.6178	480



Com a matriz de confusão, é possível percebermos que mesmo errando um bom percentual da qualidade do vinho, nos valores 5, 6 e até mesmo 7, o modelo tende a acertar a qualidade, ou errar escolhendo uma qualidade adjacente. Esse resultado só se mantém presente nos valores que possuem uma maior ocorrência

Validação Cruzada

Para confirmar a pequena melhora no f1-score, realizamos a validação cruzada:

- Antes do Grid Search:

```
# Avaliando usando validação cruzada
scores = cross_val_score(DT_clf, X_train, y_train, cv=5, scoring='f1_macro')

# Exibindo os resultados da validação cruzada
print(f'Validação cruzada F1 score (média): {scores.mean()}')
```

```
Validação cruzada F1 score (média): 0.2979018985915893
```

- Depois do Grid Search:

```
# Avaliar usando validação cruzada
scores = cross_val_score(best_dt_clf, X_train, y_train, cv=5, scoring='f1_macro'

# Exibir os resultados da validação cruzada
print(f'Validação cruzada F1 score (média): {scores.mean()}')
```

```
Validação cruzada F1 score (média): 0.3016736503342011
```

Vale ressaltar que o resultado da validação cruzada de ambos os modelos são baixas por conta da baixa ocorrência de valores para qualidade igual a 3, 4 e 8. Caso removermos esses valores, teríamos os seguintes resultados:

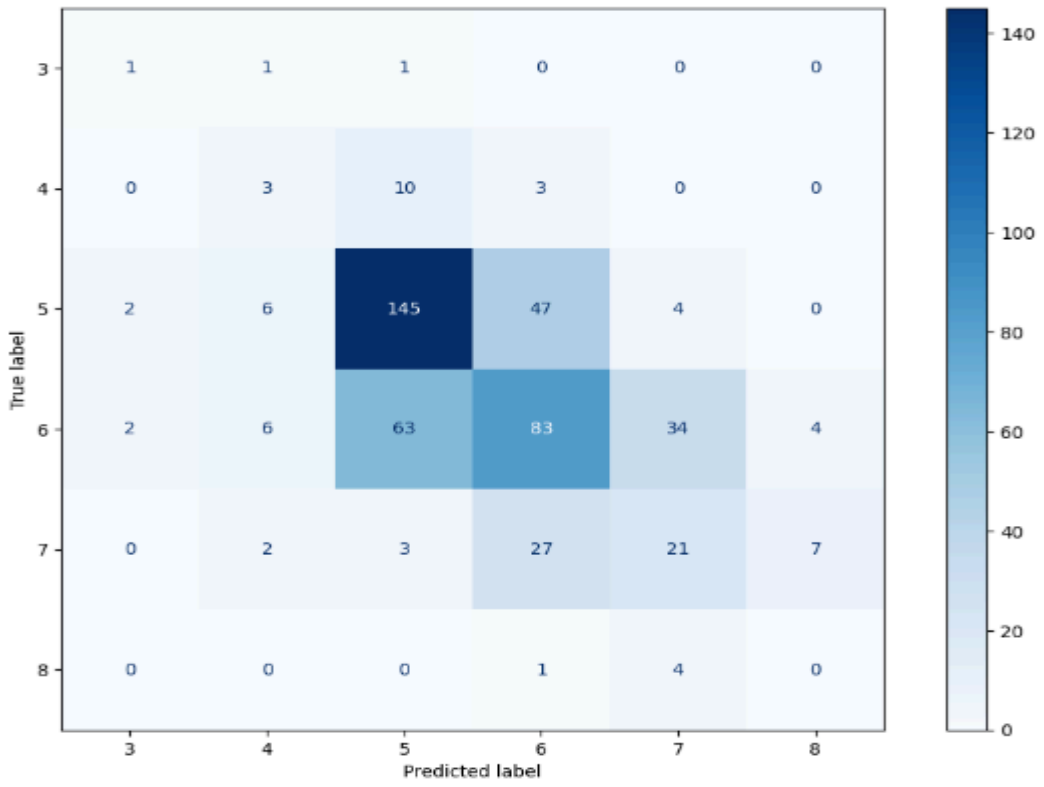
- Antes do Grid Search: 0.6166
- Depois do Grid Search: 0.6175

Bayesiano ingênuo

Desempenho

Utilizando o Bayesiano ingênuo, realizamos apenas um único treinamento visto que não há muitos hiperparâmetros para ajustar, assim já aceitando o padrão fornecido pelo sklearn. Segue o resultado do modelo gerado, seguido por sua matriz de confusão:

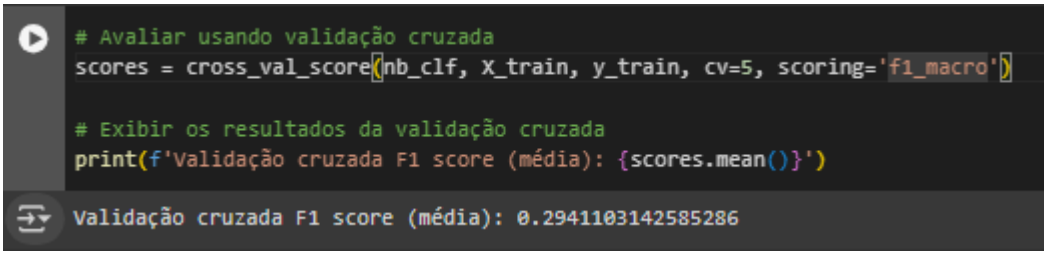
	precision	recall	f1-score	support
3	0.20	0.33	0.25	3
4	0.17	0.19	0.18	16
5	0.65	0.71	0.68	204
6	0.52	0.43	0.47	192
7	0.33	0.35	0.34	60
8	0.00	0.00	0.00	5
accuracy			0.53	480
macro avg	0.31	0.34	0.32	480
weighted avg	0.53	0.53	0.53	480



Nesse caso, também é possível visualizar que valores de qualidade iguais a 5, 6 e 7, possuem mais predições corretas, ou adjacentes.

Validação cruzada

E essa foi o resultado da validação cruzada:



Lembrando que o valor está baixo por conta da baixa ocorrência de valores 3, 4, e 8. Caso removermos eles, ficaria assim o resultado:

- Resultado de validação cruzada sem os valores 3, 4 e 8: 0.4966

Regressão logística

Desempenho Inicial

Quando utilizamos a regressão logística como classificador, realizamos inicialmente o treinamento do modelo utilizando os hiperparâmetros padrão fornecidos pela biblioteca `sklearn`. Isso nos forneceu uma linha de base para comparar as melhorias subsequentes.

Grid Search

Para otimizar o desempenho do modelo, utilizamos a técnica de Grid Search CV, que é amplamente utilizada para a otimização de hiperparâmetros em aprendizado de máquina. O Grid Search examina todas as combinações possíveis dos hiperparâmetros especificados e usa validação cruzada para avaliar o desempenho do modelo para cada combinação. No nosso caso, o Grid Search foi configurado para maximizar o F1-score macro. Os hiperparâmetros testados foram:

- `C`: [0.01, 0.1, 1, 10, 100]

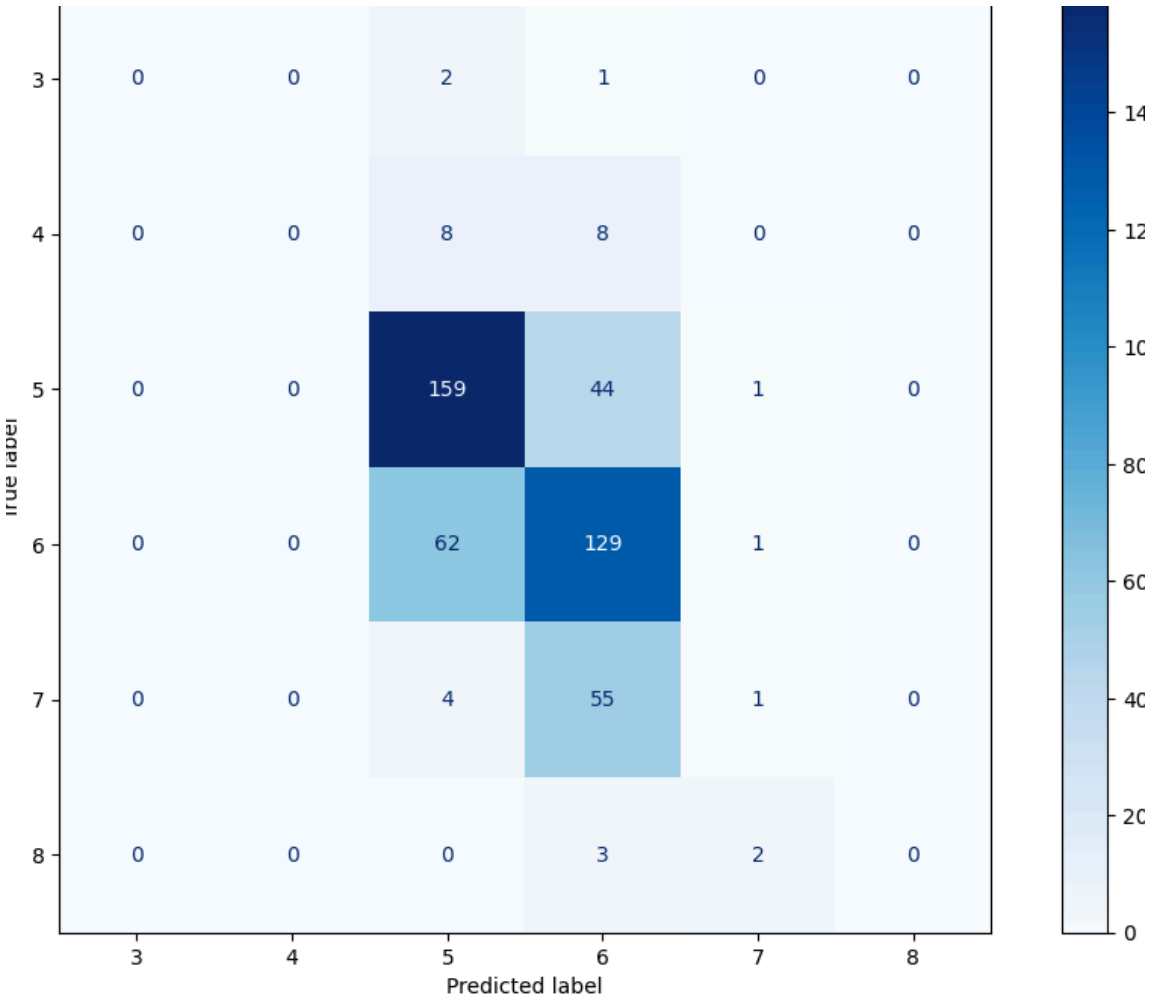
```python
from sklearn.linear_model import LogisticRegression

# Definir o modelo
lr = LogisticRegression(max_iter=1000, random_state=1)

# Definir os hiperparâmetros para busca
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}

# Configurar a validação cruzada
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
grid_search = GridSearchCV(estimator=lr, param_grid=param_grid, cv=cv, scoring='f1')
```

Desempenho Final

Após a execução do Grid Search, observamos uma melhora no F1-score, embora não significativa. O modelo final teve um desempenho ligeiramente melhor, mas os ganhos não foram substanciais. A matriz de confusão do modelo final revelou que, embora haja erros na predição da qualidade do vinho, o modelo tende a acertar ou errar por uma qualidade adjacente.

```python
# Previsões
y_pred = best_lr.predict(X_test)

# Avaliação do modelo
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred, labels=best_lr.classes_)
class_report = classification_report(y_test, y_pred)

print(class_report)
```

A matriz de confusão mostrou que, apesar dos erros na classificação, o modelo tem uma tendência a prever classes adjacentes, especialmente para as classes de maior ocorrência.



K-vizinhos

Desempenho Inicial

Inicialmente, o modelo de K-Vizinhos foi treinado usando os parâmetros padrão. Isso forneceu um ponto de partida para comparações posteriores e a identificação de melhorias.

Grid Search

Para otimizar o modelo KNN, utilizamos a técnica de Grid Search CV, configurada para maximizar o F1-score macro. A distância Euclidiana foi utilizada para definir a vizinhança, e os hiperparâmetros testados incluíram:

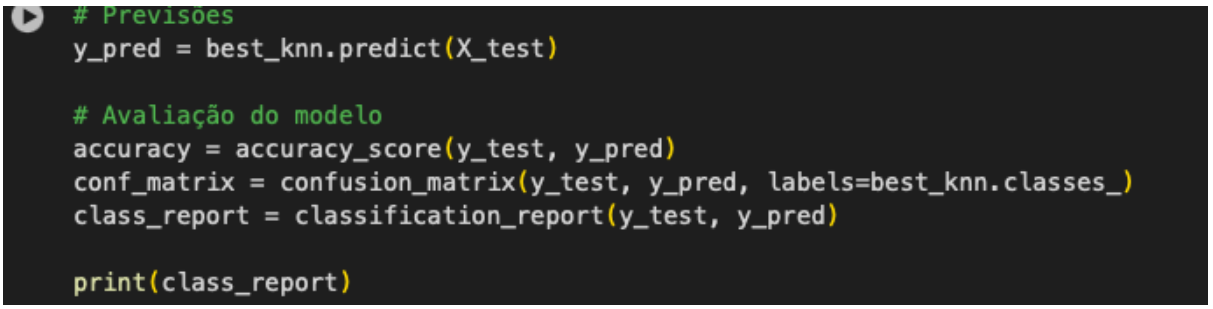
- `n_neighbors`: [1, 2, ..., 30]
- `p`: [2] (distância Euclidiana)

- 

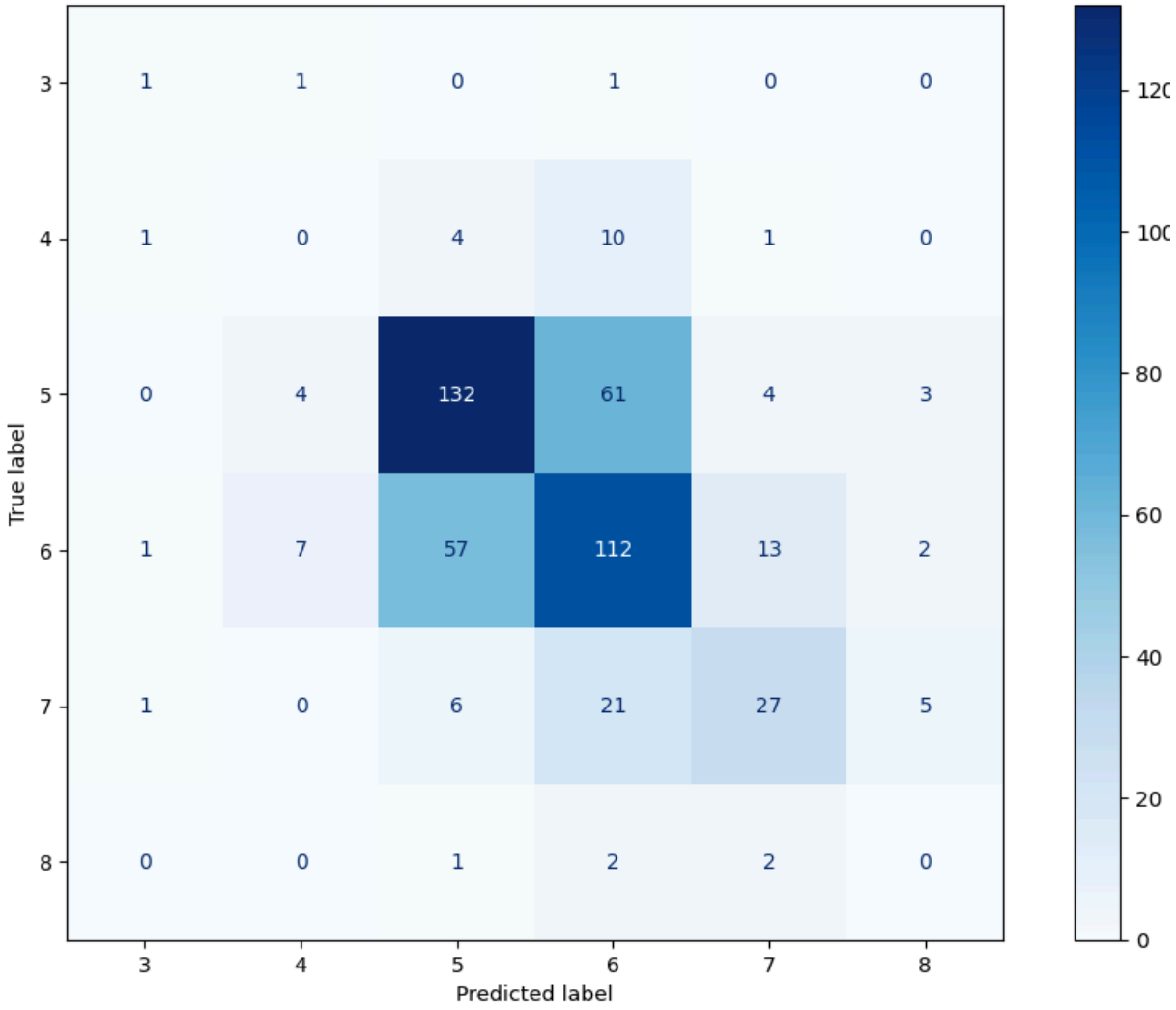
```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     # Definir o modelo
     knn = KNeighborsClassifier()

     # Definir os hiperparâmetros para busca
     param_grid = {'n_neighbors': list(range(1, 31)), 'p': [2]}  # p=2 for Euclidean distance

     # Configurar a validação cruzada
     cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
     grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=cv, scoring='f1')

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

     # Melhor modelo
     best_knn = grid_search.best_estimator_
```

Desempenho Final

Após a execução do Grid Search, o modelo KNN também apresentou uma melhora no F1-score, embora não significativa. A matriz de confusão do modelo KNN final revelou uma tendência semelhante ao modelo de regressão logística, onde o modelo tende a acertar ou errar por uma qualidade adjacente.



```
# Previsões
y_pred = best_knn.predict(X_test)

# Avaliação do modelo
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred, labels=best_knn.classes_)
class_report = classification_report(y_test, y_pred)

print(class_report)
```


True label
Predicted label
3
4
5
6
7
8
1 1 0 1 0 0
1 0 4 10 1 0
0 4 132 61 4 3
1 7 57 112 13 2
1 0 6 21 27 5
0 0 1 2 2 0
120
100
80
60
40
20
0