

Clasificación de vinos



Néstor Batista Díaz

INFORMACIÓN DEL DATASET	2
DATASET VINOS TINTOS	2
IMPORTAR DATASET	2
NORMALIZACIÓN	3
MATRIZ DE CORRELACIÓN	4
SELECCIÓN Y ENTRENAMIENTO DE MODELOS	5
BAGGING (BaggingClassifier)	5
RANDOM FOREST (RandomForestClassifier)	6
BOOSTED TREES (AdaBoostClassifier)	8
CONCLUSIÓN	9
EXPORTAR	9
IMPORTAR	9
PREDECIR CON TODOS LOS DATOS	10
DATASET VINOS BLANCOS	11
IMPORTAR DATASET	11
NORMALIZACIÓN	12
MATRIZ DE CORRELACIÓN	12
SELECCIÓN Y ENTRENAMIENTO DE MODELOS	13
BAGGING (BaggingClassifier)	14
RANDOM FOREST (RandomForestClassifier)	15
BOOSTED TREES (AdaBoostClassifier)	16
CONCLUSIÓN	17
EXPORTAR	17
IMPORTAR	17
PREDECIR CON TODOS LOS DATOS	18
CONCLUSIÓN	19
REFERENCIAS	19

INFORMACIÓN DEL DATASET

Los dos conjuntos de datos están relacionados con las variantes tintas y blancas del vino portugués "Vinho Verde". Para más detalles, consulte: <http://www.vinhoverde.pt/en/> o la referencia [Cortez et al., 2009]. Debido a cuestiones de privacidad y logística, sólo se dispone de variables fisicoquímicas (entradas) y sensoriales (la salida) (por ejemplo, no hay datos sobre tipos de uva, marca del vino, precio de venta del vino, etc.).

Estos conjuntos de datos pueden considerarse tareas de clasificación o regresión. Las clases están ordenadas y no equilibradas (por ejemplo, hay muchos más vinos normales que excelentes o malos). Podrían utilizarse algoritmos de detección de valores atípicos para detectar los pocos vinos excelentes o malos. Además, no estamos seguros de que todas las variables de entrada sean relevantes. Por eso sería interesante probar métodos de selección de características.

DATASET VINOS TINTOS

IMPORTAR DATASET

```
DATASET

df = pd.read_csv("wine+quality\\winequality-red.csv", sep=";")
df.shape
✓ 0.8s
(1599, 12)
```

```
df.describe()
```

✓ 0.1s

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

```
df.head()
```

✓ 0.5s

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

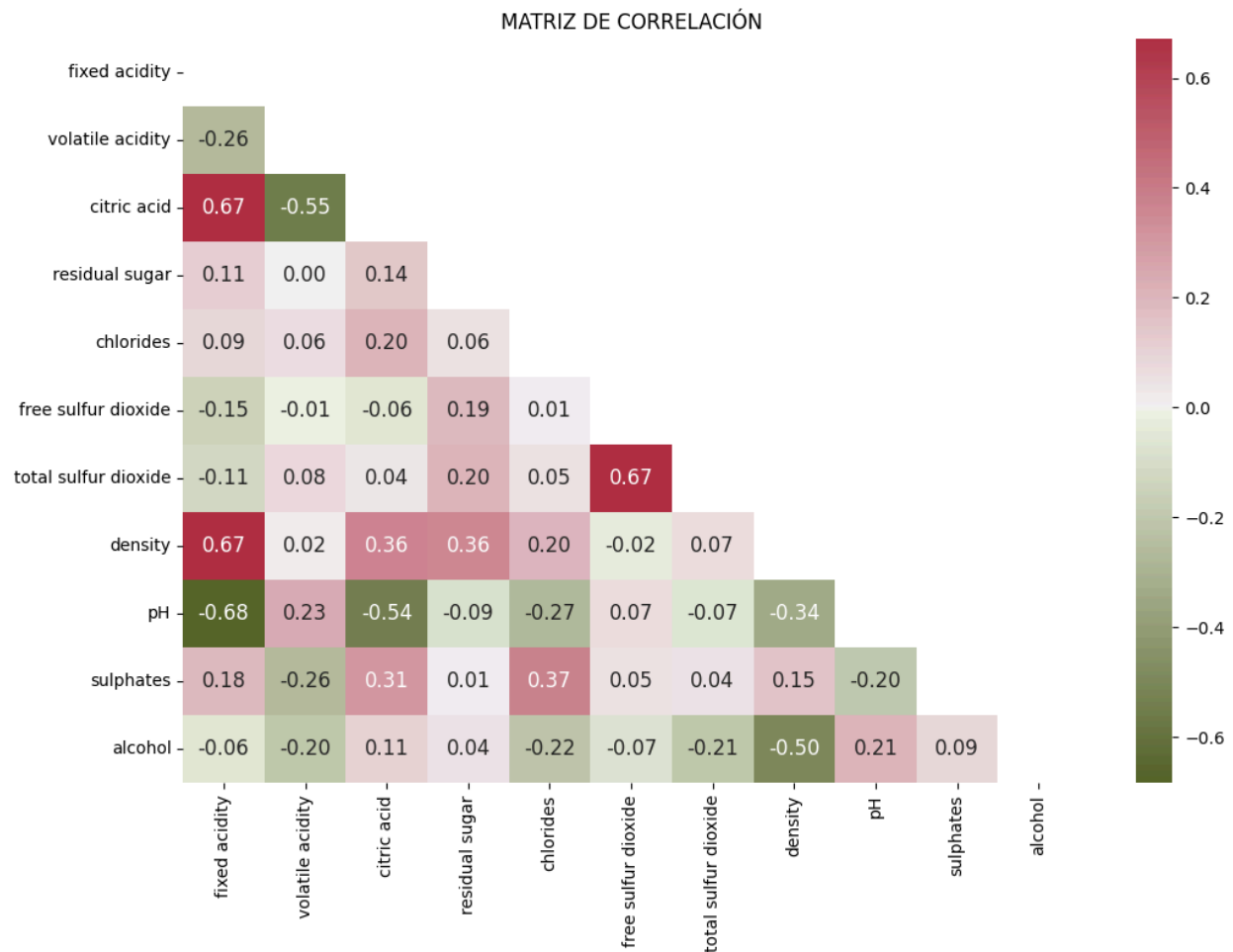
NORMALIZACIÓN

```
column_names = df.columns.values
index_to_remove = np.where(column_names == 'quality')[0]
column_names = np.delete(column_names, index_to_remove)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop('quality', axis=1))
scaled_df = pd.DataFrame(scaled_data, columns=column_names)
```

✓ 0.6s

MATRIZ DE CORRELACIÓN



Aquí podemos observar par de características que destacan como “fixed acidity”, “citric acid” y “free sulfur dioxide”.

Vamos a usar otro método con *SelectKBest* para facilitar la selección de características:

```
X = scaled_df
y = df['quality']

columns_name = df.columns.tolist()

# Crear el objeto SelectKBest con f_classif como función de puntuación
selector = SelectKBest(f_classif, k=8) # Por ejemplo, seleccionamos las 8 mejores características

# Aplicar la transformación a tus datos
X_new = selector.fit_transform(X, y)

# Obtener las características seleccionadas
selected_features = selector.get_support(indices=True)
selected_features_names = [columns_name[i] for i in selected_features]

# Imprimir las características seleccionadas
print("Características seleccionadas:", selected_features_names)
```

[400] ✓ ✓ ✓ 0.5s Python

... Características seleccionadas: ['fixed acidity', 'volatile acidity', 'citric acid', 'chlorides', 'total sulfur dioxide', 'density', 'sulphates', 'alcohol']

Estuve haciendo pruebas y con 8 características los modelos predicen mejor los resultados.

SELECCIÓN Y ENTRENAMIENTO DE MODELOS

He elegido un modelo de árbol de decisión para cada tipo de árbol.

BAGGING (BaggingClassifier)

```
# Crear un clasificador base
base_classifier = DecisionTreeClassifier()

# Crear el clasificador Bagging
bagging = BaggingClassifier(base_classifier)

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(10, 100),
    'max_samples': [0.5, 0.7, 0.9, 1.0],
    'max_features': [0.5, 0.7, 0.9, 1.0]
}

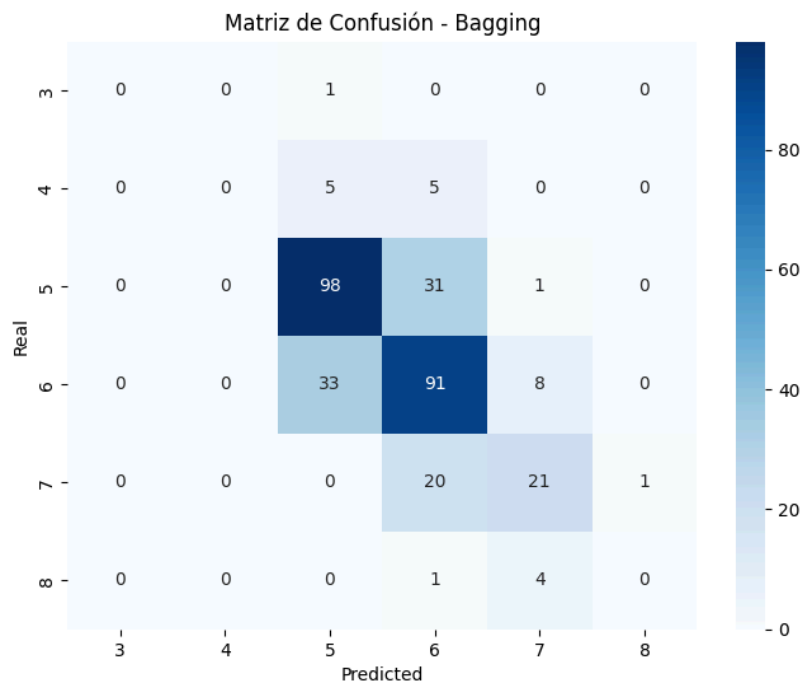
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_bagging = RandomizedSearchCV(bagging, param_distributions=param_dist, n_iter=10)

# Ajustar el modelo
random_search_bagging.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_bagging.best_params_)
print("Accuracy: ", random_search_bagging.best_score_)
```

✓ 8.2s

Mejores parámetros encontrados: {'max_features': 0.9, 'max_samples': 0.7, 'n_estimators': 71}
Accuracy: 0.6841574754901961



Parece que el modelo *BaggingClassifier* tiene buen índice de acierto y se corresponde con la matriz de confusión.

RANDOM FOREST (RandomForestClassifier)

```
# Crear el clasificador RandomForest
random_forest = RandomForestClassifier()

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
}

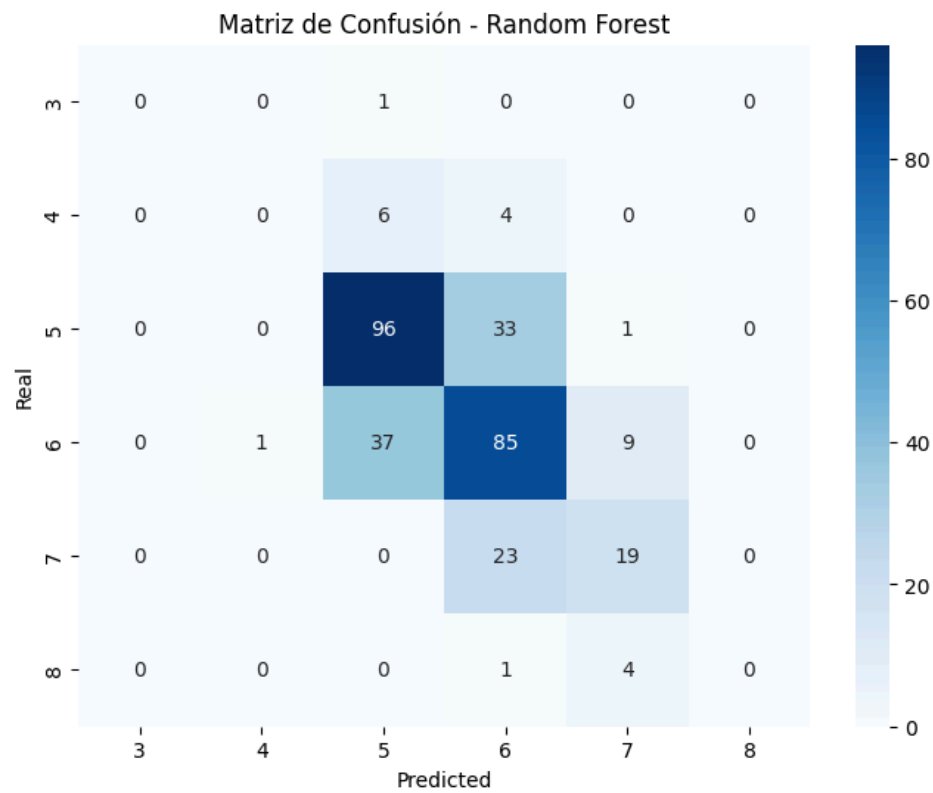
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_random_forest = RandomizedSearchCV(random_forest, param_distributions=param_dist, n_iter=10, error_score='raise')

# Ajustar el modelo
random_search_random_forest.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_random_forest.best_params_)
print("Accuracy: ", random_search_random_forest.best_score_)

✓ 1m 35s Python
```

Mejores parámetros encontrados: {'max_depth': None, 'max_features': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 703}
Accuracy: 0.6747732843137255



Parece que el modelo *RandomForestClassifier* también tiene buen índice de acierto, pero el *BaggingClassifier* es ligeramente mejor.

BOOSTED TREES (AdaBoostClassifier)

```
# Crear el clasificador AdaBoost con el algoritmo SAMME
adaboost = AdaBoostClassifier(algorithm='SAMME')

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(50, 500),
    'learning_rate': [0.01, 0.1, 1.0, 10.0]
}

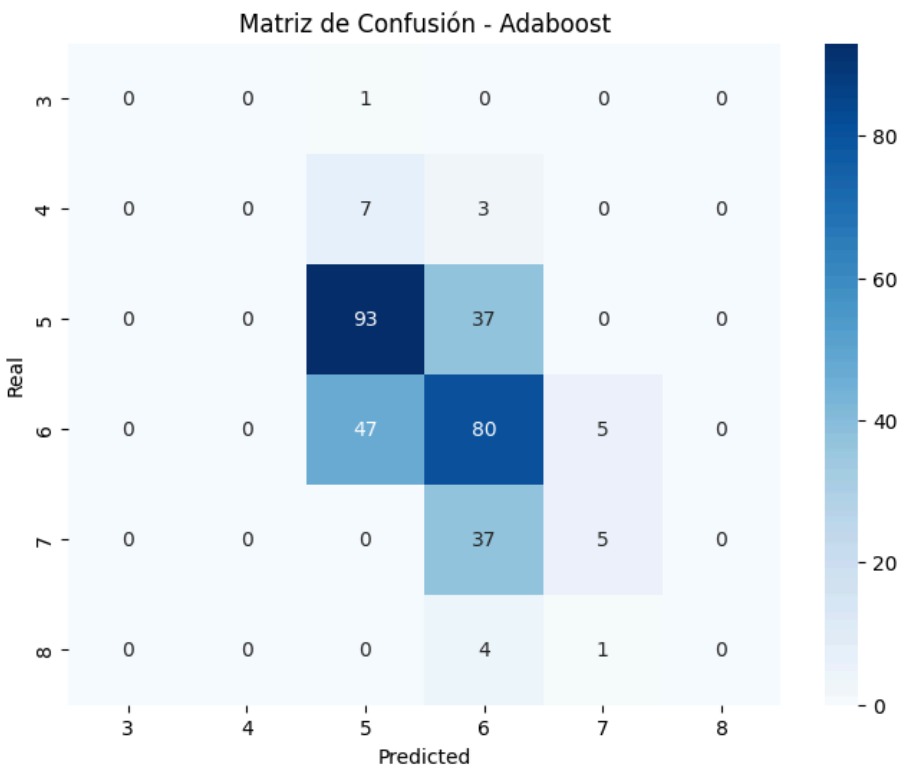
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_adaboost = RandomizedSearchCV(adaboost, param_distributions=param_dist, n_iter=10)

# Ajustar el modelo
random_search_adaboost.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_adaboost.best_params_)
print("Accuracy: ", random_search_adaboost.best_score_)

✓ 20.2s

Mejores parámetros encontrados: {'learning_rate': 0.1, 'n_estimators': 125}
Accuracy: 0.5786305147058823
```



El modelo *AdaBoostClassifier* es claramente el peor de los tres.

CONCLUSIÓN

Para este trabajo voy a escoger el modelo *BaggingClassifier* por ser el que tiene mayor índice de acierto.

EXPORTAR

```
✓ # Exportar el modelo a un fichero
  joblib.dump(random_search_bagging, 'wine_red_quality_bagging_modelo_entrenado.pkl')
] ✓ 0.1s

['wine_red_quality_bagging_modelo_entrenado.pkl']
```

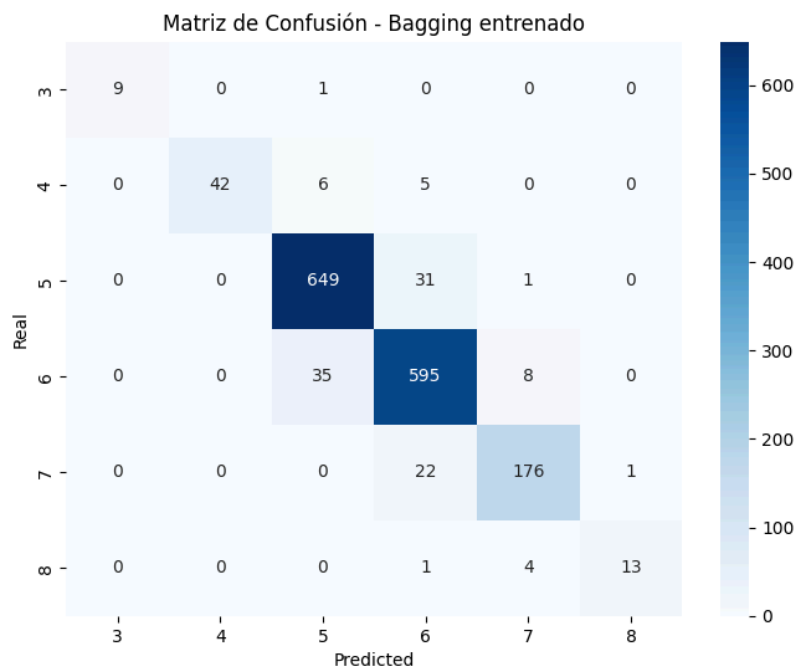
IMPORTAR

```
> ✓ modelo_bagging_entrenado = joblib.load('wine_red_quality_bagging_modelo_entrenado.pkl')
415] modelo_bagging_entrenado.score(X[selected_features_names], y)
✓ 0.2s
.. 0.9280800500312696
```

PREDECIR CON TODOS LOS DATOS

```
y_pred_modelo_bagging_entrenado = modelo_bagging_entrenado.predict(X[selected_features_names])  
✓ 0.7s
```

```
# Calcular la matriz de confusión  
cm_modelo_bagging_entrenado = confusion_matrix(y, y_pred_modelo_bagging_entrenado)  
✓ 0.4s
```



```
# Calcular la precisión de las predicciones  
accuracy = accuracy_score(y, y_pred_modelo_bagging_entrenado)  
  
print("Precisión del modelo de Bagging:", accuracy)  
✓ 0.4s  
  
Precisión del modelo de Bagging: 0.9280800500312696
```

Aquí podemos comprobar que la matriz de correlación coincide con el score dado al importar el modelo y que el *accuracy* es bastante más elevado que antes.

DATASET VINOS BLANCOS

IMPORTAR DATASET

```
df = pd.read_csv("wine+quality\\winequality-white.csv", sep=";")
df.shape
```

✓ 0.7s

(4898, 12)

```
df.describe()
```

✓ 0.7s

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000

```
df.head()
```

✓ 0.6s

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

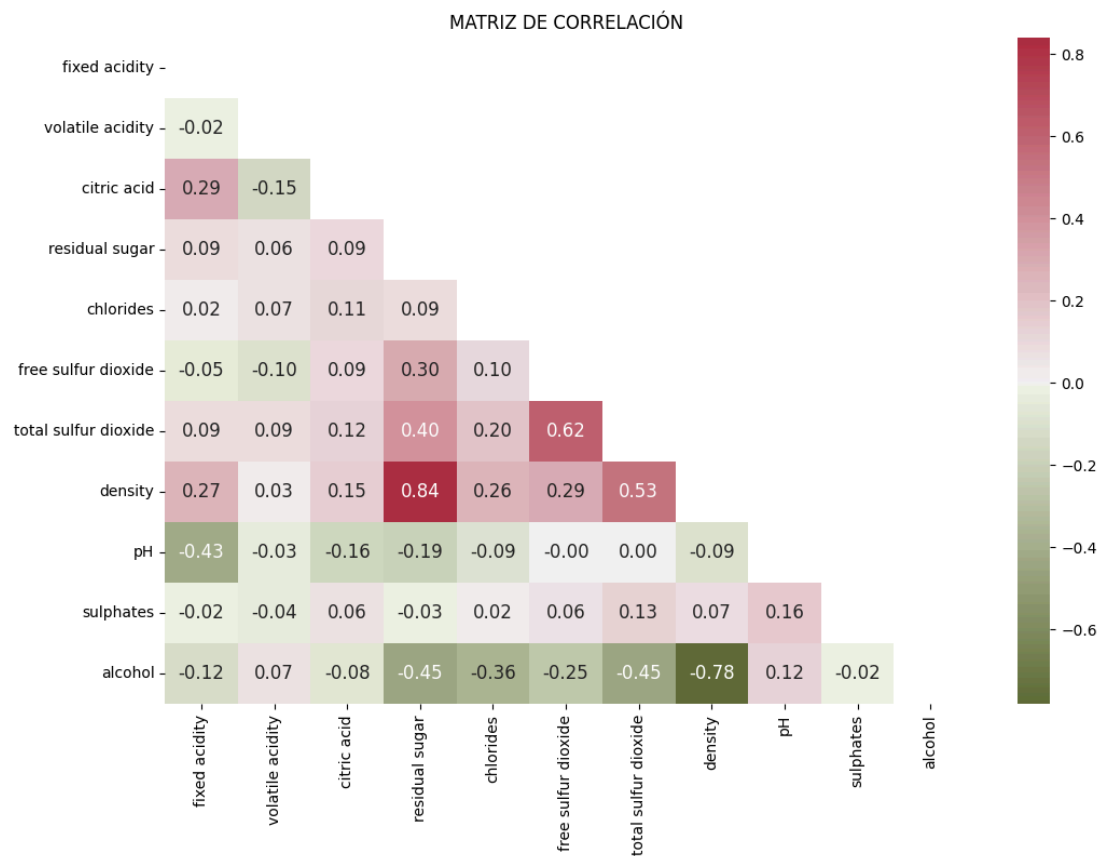
NORMALIZACIÓN

```
column_names = df.columns.values
index_to_remove = np.where(column_names == 'quality')[0]
column_names = np.delete(column_names, index_to_remove)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop('quality', axis=1))
scaled_df = pd.DataFrame(scaled_data, columns=column_names)
```

✓ 0.5s

MATRIZ DE CORRELACIÓN



Aquí podemos observar par de características que destacan como “density”, “total sulfur dioxide” y “free sulfur dioxide” y al contrario que el de los vinos tintos que destacaba más “fixed acidity”, “citric acid” y “free sulfur dioxide”.

Vamos a usar otro método con *SelectKBest* para facilitar la selección de características:

```
X = scaled_df
y = df['quality']

columns_name = df.columns.tolist()

# Crear el objeto SelectKBest con f_classif como función de puntuación
selector = SelectKBest(f_classif, k=8) # Por ejemplo, seleccionamos las 8 mejores características

# Aplicar la transformación a tus datos
X_new = selector.fit_transform(X, y)

# Obtener las características seleccionadas
selected_features = selector.get_support(indices=True)
selected_features_names = [columns_name[i] for i in selected_features]

# Imprimir las características seleccionadas
print("Características seleccionadas:", selected_features_names)
```

✓ 0.4s Python

Características seleccionadas: ['fixed acidity', 'volatile acidity', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'alcohol']

En este caso vemos mayor diferencia, en los vinos tintos las mejores características eran:

'fixed acidity', 'volatile acidity', 'citric acid', 'chlorides', 'total sulfur dioxide', 'density', 'sulphates' y 'alcohol'.

SELECCIÓN Y ENTRENAMIENTO DE MODELOS

He elegido un modelo de árbol de decisión para cada tipo de árbol.

BAGGING (BaggingClassifier)

```
# Crear un clasificador base
base_classifier = DecisionTreeClassifier()

# Crear el clasificador Bagging
bagging = BaggingClassifier(base_classifier)

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(10, 100),
    'max_samples': [0.5, 0.7, 0.9, 1.0],
    'max_features': [0.5, 0.7, 0.9, 1.0]
}

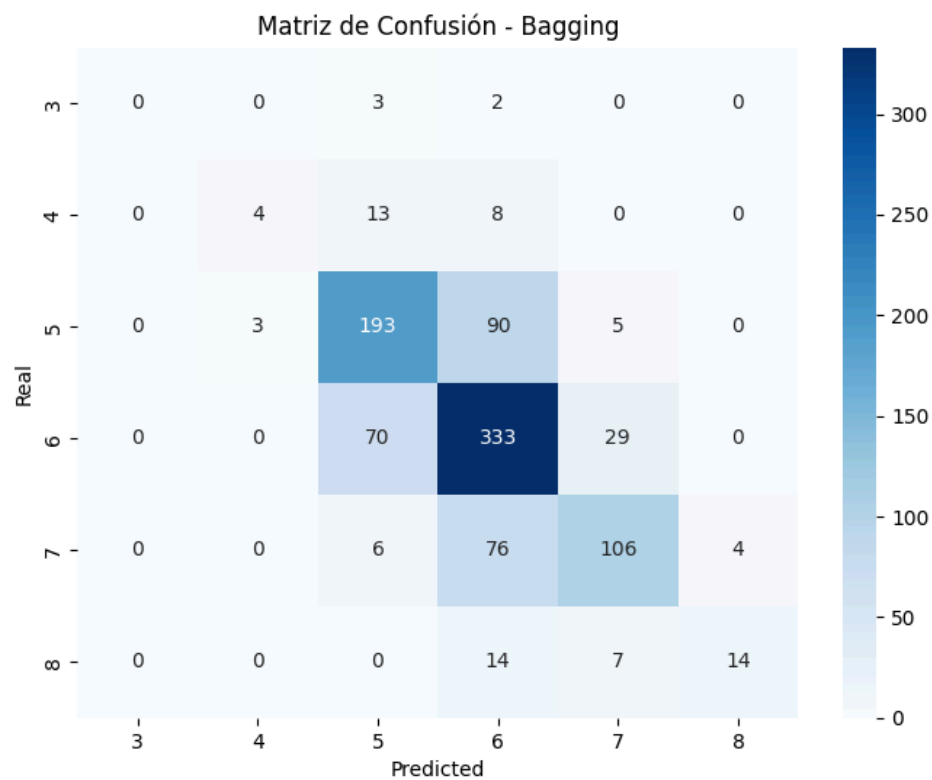
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_bagging = RandomizedSearchCV(bagging, param_distributions=param_dist, n_iter=10)

# Ajustar el modelo
random_search_bagging.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_bagging.best_params_)
print("Accuracy: ", random_search_bagging.best_score_)

✓ 21.1s

Mejores parámetros encontrados: {'max_features': 0.5, 'max_samples': 0.9, 'n_estimators': 46}
Accuracy: 0.6510956681523183
```



Parece que el modelo *BaggingClassifier* tiene buen índice de acierto y se corresponde con la matriz de confusión, pero no es tan buena como en los vinos tintos.

RANDOM FOREST (RandomForestClassifier)

```
# Crear el clasificador RandomForest
random_forest = RandomForestClassifier()

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', None]
}

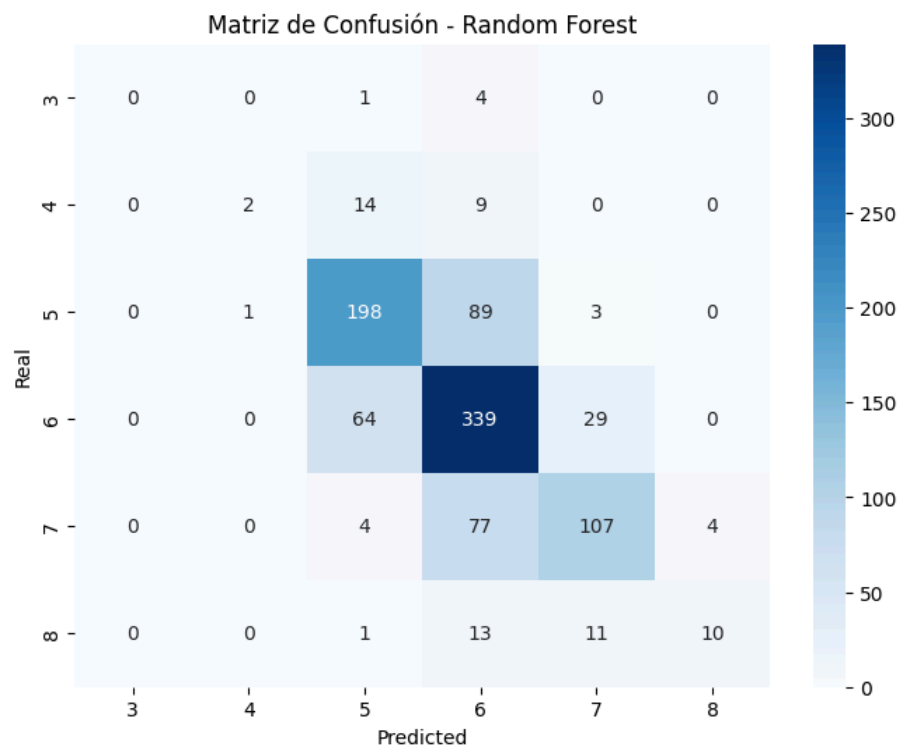
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_random_forest = RandomizedSearchCV(random_forest, param_distributions=param_dist, n_iter=10, error_score='raise')

# Ajustar el modelo
random_search_random_forest.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_random_forest.best_params_)
print("Accuracy: ", random_search_random_forest.best_score_)

✓ 1m 49.6s

Mejores parámetros encontrados: {'max_depth': 50, 'max_features': 'log2', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 982}
Accuracy: 0.6475190267677953
```



Parece que el modelo *RandomForestClassifier* también tiene buen índice de acierto, pero el *BaggingClassifier* es ligeramente mejor. También es algo peor comparado con el modelo entrenado con los vinos tintos.

BOOSTED TREES (AdaBoostClassifier)

```
# Crear el clasificador AdaBoost con el algoritmo SAMME
adaboost = AdaBoostClassifier(algorithm='SAMME')

# Definir los parámetros que deseas ajustar
param_dist = {
    'n_estimators': randint(50, 500),
    'learning_rate': [0.01, 0.1, 1.0, 10.0]
}

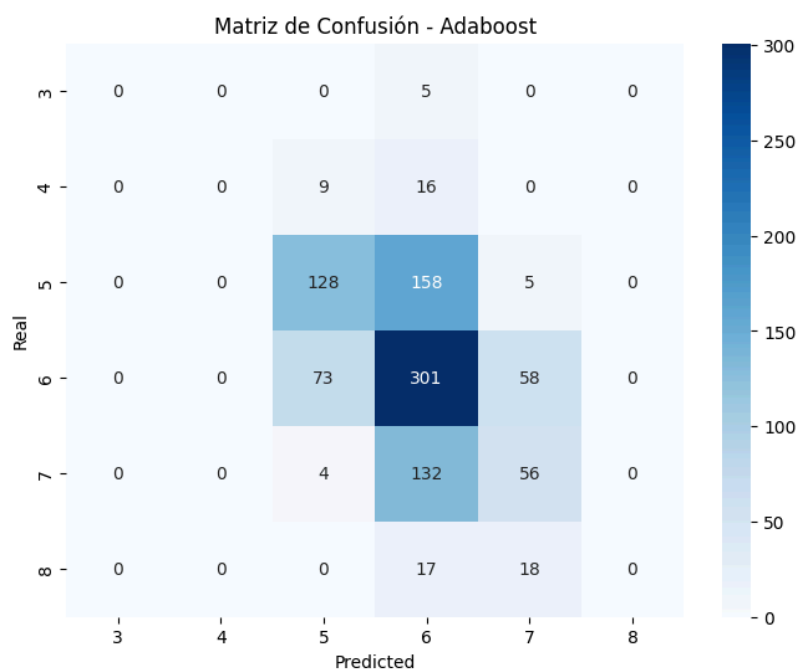
# Realizar la búsqueda aleatoria de hiperparámetros
random_search_adaboost = RandomizedSearchCV(adaboost, param_distributions=param_dist, n_iter=10)

# Ajustar el modelo
random_search_adaboost.fit(X_train, y_train)

# Mostrar los mejores parámetros encontrados
print("Mejores parámetros encontrados:", random_search_adaboost.best_params_)
print("Accuracy: ", random_search_adaboost.best_score_)

✓ 32.3s

Mejores parámetros encontrados: {'learning_rate': 0.01, 'n_estimators': 428}
Accuracy: 0.51530807725389
```



El modelo *AdaBoostClassifier* es claramente el que más se ve afectado en comparación al entrenamiento con el dataset de los vinos tintos

CONCLUSIÓN

Para este trabajo voy a escoger el modelo *BaggingClassifier* por ser el que tiene mayor índice de acierto.

EXPORTAR

```
# Exportar el modelo a un fichero
joblib.dump(random_search_bagging, 'wine_white_quality_bagging_modelo_entrenado.pkl')
✓ 0.8s
['wine_white_quality_bagging_modelo_entrenado.pkl']
```

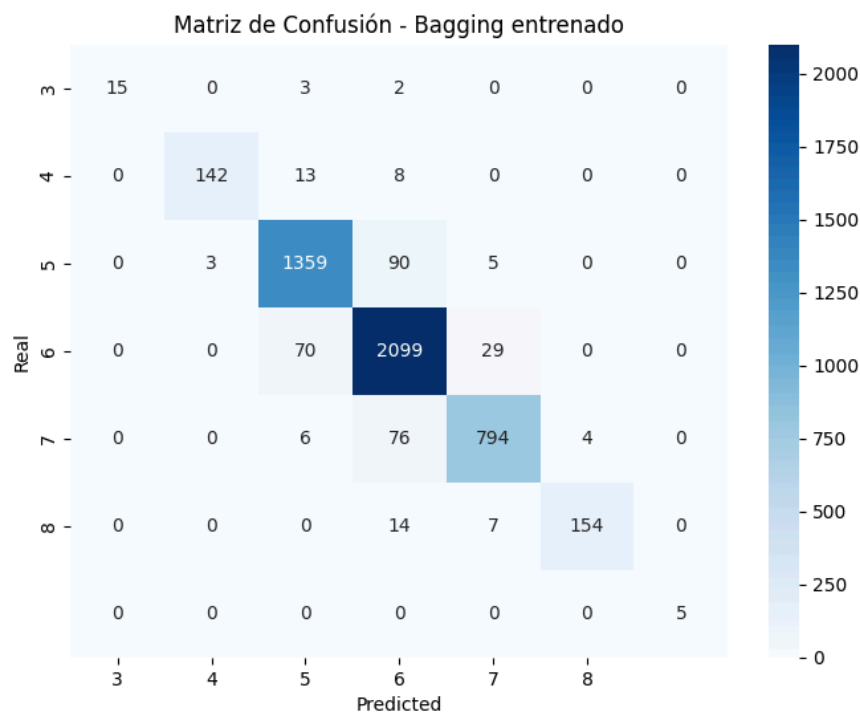
IMPORTAR

```
modelo_bagging_entrenado = joblib.load('wine_white_quality_bagging_modelo_entrenado.pkl')
modelo_bagging_entrenado.score(X[selected_features_names], y)
✓ 0.2s
0.9326255614536546
```


PREDECIR CON TODOS LOS DATOS

```
y_pred_modelo_bagging_entrenado = modelo_bagging_entrenado.predict(X[selected_features_names])  
✓ 0.8s
```

```
# Calcular la matriz de confusión  
cm_modelo_bagging_entrenado = confusion_matrix(y, y_pred_modelo_bagging_entrenado)  
✓ 0.4s
```



```
# Calcular la precisión de las predicciones  
accuracy = accuracy_score(y, y_pred_modelo_bagging_entrenado)  
print("Precisión del modelo de Bagging:", accuracy)  
✓ 0.3s  
Precisión del modelo de Bagging: 0.9326255614536546
```



Aquí podemos comprobar que la matriz de correlación coincide con el score dado al importar el modelo y que el *accuracy* es bastante más elevado que antes. También comprobamos que pese a que los modelos daban un índice menor de acierto ahora a superado ligeramente al modelo entrenado de los vinos tintos.

CONCLUSIÓN

No hay que obsesionarse tanto con el accuracy dado de cada modelo, simplemente ceñirse a elegir el mejor y cuando se entrene se verá lo potente que puede ser.

REFERENCIAS

- <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>
- https://colab.research.google.com/drive/1vt1_mUA_IsUHwEMYBuorX0SpvtOHUcld?usp=sharing
- <https://github.com/Nestorbd/prediction-Mohs-Hardness>
- <https://github.com/Nestorbd/Clasificacion-de-vinos>