Reconocimiento de Patrones (ML) - T02

ALEJANDRO ZARATE MACIAS

09 de Febrero 2026

Abstract

Esta tarea continúa el estudio del aprendizaje supervisado enfocándose en problemas de clasificación. Se implementan y comparan modelos como KNN y Regresión Logística en escenarios de clasificación binaria y multiclase, utilizando métricas adecuadas como F1-score y matrices de confusión para evaluar su desempeño y capacidad de generalización.

1 Problema 1

1.1 Enunciado

Realiza una pequeña investigación para responder a la siguiente pregunta: ¿por qué la función MSE, que antes se utilizaba como función de costo al realizar regresión, ya no se usa comúnmente como función de costo en el escenario de clasificación? Mantén tu respuesta simple.

1.2 Análisis

El Mean Squared Error (MSE) ha perdido popularidad en problemas de clasificación principalmente por dos razones fundamentales:

1. Naturaleza del error: MSE calcula el error utilizando valores continuos, lo cual es ideal para modelos de regresión donde el objetivo es predecir un valor continuo. Este enfoque permite ajustes graduales y precisos basados en el error calculado. Sin embargo, en clasificación binaria, el modelo solo puede predecir dos valores discretos (0 o 1), por lo que el cálculo continuo del error de MSE no resulta tan útil para realizar ajustes precisos del modelo. La discrepancia entre la naturaleza continua del error y la naturaleza discreta de las predicciones crea una incompatibilidad fundamental.

2. Aparición de funciones más apropiadas: La llegada de funciones de pérdida basadas en logaritmos, como la entropía cruzada (cross-entropy), marcó un cambio significativo. Estas funciones se adaptan mejor a problemas de clasificación porque penalizan de manera más efectiva los errores cuando el modelo está muy seguro de su predicción pero esta resulta incorrecta. Esta característica es crucial en clasificación, ya que queremos que el modelo sea

castigado más severamente por predicciones confiadas pero erróneas, algo que MSE no logra de manera eficiente.

1.3 Conclusión

En resumen, MSE no proporciona resultados tan efectivos como otras funciones de pérdida en problemas de clasificación debido a su naturaleza continua que no se alinea bien con las salidas discretas de clasificación. Por esta razón, funciones como cross-entropy han tomado su lugar como el estándar para este tipo de aplicaciones.

2 Problema 2

2.1 Enunciado

Lea el capítulo 4.5 de [5], centrado en los modelos KNN y de regresión logística. ¿Cómo se comparan? Escriba sus conclusiones.

2.2 Análisis

La Regresión Logística y K-Nearest Neighbors (KNN) son dos enfoques populares para clasificación que, aunque resuelven la misma tarea, lo hacen desde perspectivas muy diferentes:

Diferencias fundamentales:

Regresión Logística es un modelo paramétrico que busca encontrar una función que se ajuste a los datos de entrenamiento y pueda predecir la probabilidad de que una observación pertenezca a una clase determinada. Este modelo aprende parámetros específicos durante el entrenamiento que definen la frontera de decisión.

KNN, por otro lado, es un modelo no paramétrico basado en la idea de que observaciones similares tienden a pertenecer a la misma clase. Para clasificar una nueva observación, KNN simplemente busca las k observaciones más cercanas en el espacio de características y asigna la clase mayoritaria entre esos vecinos. No hace suposiciones sobre la forma de la función subyacente y utiliza directamente los datos de entrenamiento para hacer predicciones.

Resultados experimentales del capítulo:

El texto presenta varios experimentos comparativos que revelan patrones importantes:

- **Frontera de decisión lineal:** La regresión logística suele superar a KNN, ya que este último paga un precio innecesario en varianza al no aprovechar la linealidad de la frontera.
- **Frontera moderadamente no lineal:** Métodos más flexibles como KNN pueden ofrecer mejores resultados al adaptarse mejor a la complejidad de la frontera.

- **Frontera altamente no lineal:** KNN con una selección adecuada de K puede superar a los métodos lineales, aunque sigue siendo sensible a la elección de este hiperparámetro de suavizamiento.

2.3 Conclusión

No existe un modelo claramente superior entre Regresión Logística y KNN. La elección apropiada depende del tipo de problema que se quiera resolver, la naturaleza de los datos disponibles y, especialmente, la complejidad de la frontera de decisión. Es fundamental evaluar ambos modelos en el contexto específico del problema para determinar cuál es el más adecuado. En general, si esperamos una relación lineal, la regresión logística será más eficiente; si la frontera es compleja y tenemos suficientes datos, KNN puede ser más apropiado.

3 Problema 3

3.1 Enunciado

Considere el conjunto de datos de [1]. Cree un script con sklearn para resolver el problema de clasificación binaria asociado con determinar si un caballo determinado sobrevivirá o no. Utilice el modelo KNN de sklearn. Anote todas las suposiciones y operaciones de preprocesamiento de datos que realice. Incorpore la puntuación F1 para explicar sus hallazgos [2].

3.2 Metodología

Para abordar este problema de clasificación binaria se siguió un enfoque sistemático dividido en cuatro etapas:

1. Carga de datos desde Kaggle
2. Exploración inicial
3. Preprocesamiento de datos
4. Entrenamiento del modelo y evaluaciones

La carga de datos se realizó empleando funciones de la librería `kagglehub` disponibles en Python. Posteriormente se procedió a una exploración inicial para entender la estructura de los datos, identificar valores faltantes y analizar la distribución de las clases.

Del análisis inicial se identificó que el dataset contiene 299 observaciones y 28 variables, presentando desafíos importantes: más de la mitad de las columnas tienen valores faltantes y gran parte son categóricas, lo que requiere un preprocesamiento adecuado. La variable objetivo `outcome` tiene tres valores: *lived* (170 casos), *died* (77 casos) y *euthanized* (44 casos). Para convertir esto en un problema binario, se asignó 1 a “lived” y 0 a “died” o “euthanized”. De las 28

columnas, muchas contienen información vital sobre la salud del caballo y signos vitales; sin embargo, se identificaron columnas que no aportan información predictiva relevante como `hospital_number`, `lesion_1`, `lesion_2`, `lesion_3` y `cp_data`, por lo que fueron eliminadas.

Una vez realizada la exploración inicial, se procedió al preprocesamiento de datos que incluye las siguientes operaciones:

Variable objetivo: Se generó una columna llamada `outcome_binary` a partir de la columna `outcome`:

```
# Convertir outcome a binario: lived=1, died/euthanized=0
df['outcome_binary'] = (df['outcome'] == 'lived').astype(int
    )
```

Manejo de valores faltantes: Se utilizó `SimpleImputer` de sklearn, que permite imputar los valores faltantes utilizando la estrategia de la mediana para las variables numéricas y la moda para las variables categóricas:

```
imputer_num = SimpleImputer(strategy='median')
imputer_cat = SimpleImputer(strategy='most_frequent')
X[numeric_cols] = imputer_num.fit_transform(X[numeric_cols])
X[categorical_cols] = imputer_cat.fit_transform(X[
    categorical_cols])
```

Codificación de variables categóricas: Se empleó la función `get_dummies` de pandas para convertir las variables categóricas en variables dummy, lo que permite que el modelo KNN pueda procesar esta información de manera adecuada.

Separación de datos: Se dividió el conjunto de datos en entrenamiento (80%) y prueba (20%) utilizando `train_test_split` de sklearn, con `stratify=y` para mantener la proporción de clases y la semilla 14 para garantizar la reproducibilidad.

Normalización: Dado que el modelo KNN es sensible a la escala de las características, se aplicó normalización utilizando `StandardScaler` de sklearn. Este paso es crucial ya que KNN utiliza distancias para determinar los vecinos más cercanos.

Con todo esto se obtuvo un conjunto de datos limpio y preparado para entrenar el modelo KNN, con 239 observaciones y 41 características para entrenamiento, y 60 observaciones para prueba.

Finalmente, se entrenó el modelo KNN utilizando la clase `KNeighborsClassifier` de sklearn. Se implementó una clase personalizada que encapsula el modelo y facilita el entrenamiento y evaluación:

```
class KNNModel:
    def __init__(self, n_neighbors=3):
        self.n_neighbors = n_neighbors
        self.model = KNeighborsClassifier(n_neighbors=self.
            n_neighbors)
        self.scaler = StandardScaler()
```

```
    def fit(self, X_train, y_train):
        # Entrenamiento
        X_train_scaled = self.scaler.fit_transform(X_train)
        self.model.fit(X_train_scaled, y_train)
        # Metricas
        y_train_pred = self.model.predict(X_train_scaled)
        train_acc = self.model.score(X_train_scaled, y_train
            )
        train_f1 = f1_score(y_train, y_train_pred)
        return train_acc, train_f1

    def predict(self, X_test, y_test):
        # Prediccion
        X_test_scaled = self.scaler.transform(X_test)
        y_pred = self.model.predict(X_test_scaled)
        # Metricas
        test_acc = self.model.score(X_test_scaled, y_test)
        test_f1 = f1_score(y_test, y_pred)
        return test_acc, test_f1
```

Se comenzó con un valor de $k = 5$, el cual es el valor por defecto que utiliza la función de acuerdo a la documentación oficial, con el objetivo de evaluar cómo se desempeña el modelo en su configuración base. Adicionalmente, se realizaron pruebas con distintos valores de k (desde 1 hasta 100) para evaluar el desempeño del modelo y encontrar el valor óptimo según el F1 Score.

3.3 Resultados

Los resultados obtenidos del modelo KNN base con $k = 5$ fueron los siguientes:

- **Train:** Accuracy = 0.7992, F1 Score = 0.8367
- **Test:** Accuracy = 0.8000, F1 Score = 0.8378

Estos resultados iniciales muestran un desempeño bastante aceptable en ambos conjuntos de entrenamiento y prueba, con valores similares que sugieren buena generalización del modelo.

La búsqueda exhaustiva probando valores de k desde 1 hasta 100 reveló que $k = 16$ es el valor óptimo:

- **Mejor k según Accuracy:** 16 (Accuracy: 0.8667)
- **Mejor k según F1 Score:** 16 (F1 Score: 0.8919)

La Figura 1 muestra la evolución del desempeño del modelo en función de k. Se observa que valores muy pequeños de k (cercanos a 1) producen sobreajuste con alta varianza, mientras que valores muy grandes resultan en subajuste al suavizar excesivamente la frontera de decisión. El punto óptimo en $k = 16$ representa el mejor balance entre sesgo y varianza para este conjunto de datos específico.

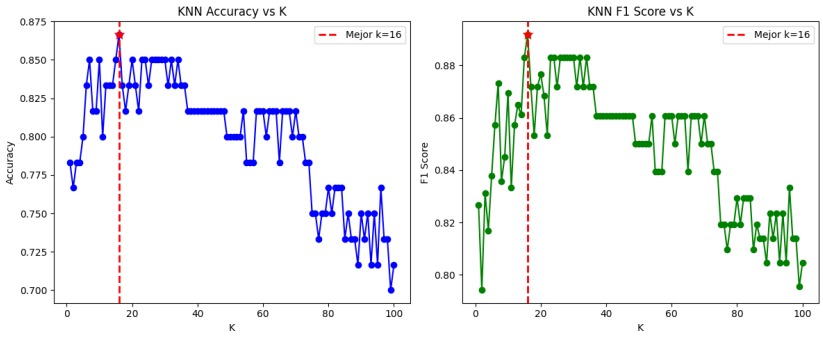

Figure 1: Evolución del Accuracy y F1 Score en función del hiperparámetro k para el modelo KNN. La estrella roja indica el valor óptimo $k = 16$.

3.4 Discusión

La mejora del F1 Score de 0.8378 a 0.8919 (incremento del 6.5%) al optimizar k demuestra la importancia de la selección adecuada de hiperparámetros. El F1 Score es particularmente relevante en este problema debido al desbalance de clases (170 supervivientes vs 121 no supervivientes), ya que considera tanto precisión como recall, evitando sesgos hacia la clase mayoritaria.

El valor óptimo $k = 16$ sugiere que considerar aproximadamente 16 vecinos cercanos proporciona suficiente información para decisiones robustas sin incluir ruido de observaciones muy distantes. Valores menores ($k < 5$) muestran mayor varianza y sensibilidad a outliers, mientras que valores mayores ($k > 30$) tienden a suavizar demasiado la frontera de decisión.

Es notable que tanto Accuracy como F1 Score coinciden en identificar $k = 16$ como óptimo, lo cual refuerza la confianza en esta configuración. Sin embargo, el F1 Score proporciona una evaluación más completa del desempeño en este contexto de clases desbalanceadas.

3.5 Conclusión

El modelo KNN demostró ser muy efectivo para predecir la supervivencia de caballos, alcanzando un F1 Score de 0.8919 y Accuracy de 0.8667 en el conjunto de prueba con $k = 16$.

El valor óptimo de k encontrado fue 16, lo que sugiere que este valor es el más adecuado para este conjunto de datos específico. La mejora de 0.8378 a 0.8919 en el F1 Score representa un incremento del 6.5%, lo que demuestra una mejora significativa en la capacidad del modelo para predecir correctamente la supervivencia de los caballos comparado con la configuración por defecto.

Es importante destacar que, comparado con el accuracy tradicional, el F1 Score proporciona una mejor perspectiva del desempeño del modelo, especialmente en este contexto donde existe desbalance de clases (170 supervivientes vs 121 no supervivientes).

4 Problema 4

4.1 Enunciado

Repita el problema 3, pero utilice el modelo de regresión logística con sklearn. Compare los resultados con los anteriores. Además, anote los hiperparámetros de optimización que eligió y explique por qué.

4.2 Metodología

Para este problema de clasificación con regresión logística se siguió un proceso similar al Problema 3, pero con diferencias importantes en la fase de entrenamiento. Las etapas principales fueron:

1. Preprocesamiento de datos (mismo proceso del Problema 3)
2. Entrenamiento con Regresión Logística
3. Exploración sistemática de hiperparámetros
4. Evaluación con configuración óptima

Dado que el análisis exploratorio y el preprocesamiento se realizaron exhaustivamente en el Problema 3, se procedió directamente con la misma transformación de datos: conversión de variable objetivo a binaria, manejo de valores faltantes, codificación one-hot de categóricas, división train-test (80/20) con estratificación, y normalización con `StandardScaler`.

La diferencia principal radica en el modelo utilizado. Como se discutió en el Problema 2, la regresión logística es un modelo paramétrico que aprende parámetros específicos durante el entrenamiento. Se implementó una clase personalizada para encapsular el modelo:

```
class LRModel:
    def __init__(self, C=1.0, max_iter=1000, class_weight=
        None,
                 solver='lbfgs', random_state=14):
        self.C = C
        self.max_iter = max_iter
        self.class_weight = class_weight
        self.solver = solver
        self.model = LogisticRegression(
            random_state=random_state,
            C=C,
            max_iter=max_iter,
            class_weight=class_weight,
            solver=solver
        )
        self.scaler = StandardScaler()

    def fit(self, X_train, y_train):
```

```
        X_train_scaled = self.scaler.fit_transform(X_train)
        self.model.fit(X_train_scaled, y_train)
        y_train_pred = self.model.predict(X_train_scaled)
        train_acc = self.model.score(X_train_scaled, y_train
            )
        train_f1 = f1_score(y_train, y_train_pred)
        return train_acc, train_f1

    def predict(self, X_test, y_test):
        X_test_scaled = self.scaler.transform(X_test)
        y_pred = self.model.predict(X_test_scaled)
        test_acc = self.model.score(X_test_scaled, y_test)
        test_f1 = f1_score(y_test, y_pred)
        return test_acc, test_f1
```

Debido a la naturaleza paramétrica de la regresión logística, se realizaron pruebas sistemáticas con diferentes hiperparámetros para encontrar la mejor configuración. Los hiperparámetros explorados fueron:

C (Regularización): Controla la regularización del modelo. Valores más pequeños implican mayor regularización. Se probaron valores [0.01, 0.1, 1, 10, 100] para encontrar el balance óptimo entre sesgo y varianza.

class_weight: Maneja el desbalance de clases. Se comparó la configuración por defecto (`None`) con `'balanced'`, que ajusta automáticamente los pesos inversamente proporcionales a las frecuencias de clase.

solver: Algoritmo de optimización. Se evaluaron cinco solvers disponibles en sklearn: `'lbfgs'`, `'liblinear'`, `'newton-cg'`, `'sag'` y `'saga'`, cada uno con características y eficiencias diferentes.

Para todos los experimentos se mantuvieron constantes `random_state=14` (reproducibilidad) y `max_iter=1000` (garantizar convergencia).

4.3 Resultados

El modelo de regresión logística base con configuración por defecto obtuvo:

- **Train:** Accuracy = 0.8159, F1 Score = 0.8472
- **Test:** Accuracy = 0.7667, F1 Score = 0.8205

La exploración del hiperparámetro C reveló resultados interesantes:

C	Train F1	Test F1	Test Acc
0.01	0.8066	**0.8571**	0.8167
0.1	0.8339	0.8421	0.8000
1	0.8472	0.8205	0.7667
10	0.8512	0.8205	0.7667
100	0.8512	0.8205	0.7667

El mejor valor fue **C=0.01**, indicando que mayor regularización beneficia la generalización.

La comparación de `class_weight` mostró:

Configuración	Train Acc	Train F1	Test Acc	Test F1
Sin balanceo	0.8159	0.8472	0.7667	0.8205
Con balanced	0.8075	0.8296	0.8000	**0.8333**

El balanceo de clases mejoró el F1 Score en el conjunto de prueba.
La evaluación de solvers identificó a **liblinear** como el mejor:

Solver	Test Acc	Test F1
lbfgs	0.7667	0.8205
liblinear	0.7833	**0.8312**
newton-cg	0.7667	0.8205
sag	0.7667	0.8205
saga	0.7667	0.8205

Finalmente, la combinación óptima (C=0.01, class_weight='balanced', solver='liblinear') produjo:

- **Train:** Accuracy = 0.7615, F1 Score = 0.7881
- **Test:** Accuracy = 0.8167, F1 Score = 0.8406

4.4 Discusión

Los resultados revelan varios aspectos importantes sobre el comportamiento de la regresión logística en este problema:

Regularización (C): El valor óptimo C=0.01 indica que este dataset se beneficia significativamente de una fuerte regularización. Esto previene el sobreajuste y mejora la generalización, como se evidencia en el mejor F1 Score de 0.8571 en prueba. Valores mayores de C permitieron que el modelo se ajustara demasiado a los datos de entrenamiento, reduciendo su capacidad predictiva.

Balanceo de clases: El uso de `class_weight='balanced'` mejoró el desempeño al compensar el desbalance entre las clases superviviente/no superviviente. Esto es consistente con las observaciones del Problema 3 sobre la importancia de manejar adecuadamente el desbalance.

Solver: El mejor desempeño de `liblinear` puede atribuirse a que este solver está optimizado para problemas de clasificación binaria y maneja bien datasets de tamaño pequeño a mediano como el presente (299 observaciones).

Comparación KNN vs Regresión Logística: El modelo KNN alcanzó un F1 Score de 0.8919 mientras que la regresión logística logró 0.8406. La diferencia sugiere que la frontera de decisión en este problema tiene componentes no lineales que KNN captura mejor. Sin embargo, ambos modelos son competitivos y adecuados para este problema.

Un hallazgo interesante es que el modelo final con configuración óptima mostró menor F1 en entrenamiento (0.7881) que en prueba (0.8406), lo cual es inusual pero puede indicar que la fuerte regularización y el balanceo de clases ayudaron específicamente con las características del conjunto de prueba.

4.5 Conclusión

La regresión logística demostró ser un modelo efectivo para predecir la supervivencia de caballos, alcanzando un F1 Score de 0.8406 en el conjunto de prueba con la configuración óptima.

Los hiperparámetros óptimos encontrados fueron:

- **C = 0.01:** Regularización fuerte para prevenir sobreajuste
- **class_weight = 'balanced':** Manejo apropiado del desbalance de clases
- **solver = 'liblinear':** Algoritmo de optimización más eficiente para este dataset

Comparado con el modelo KNN del Problema 3 (F1 = 0.8919), la regresión logística obtuvo un desempeño ligeramente inferior (F1 = 0.8406), con una diferencia de aproximadamente 5%. Esta diferencia sugiere que:

1. Ambos modelos son viables para este problema, con buen poder predictivo.
2. KNN tiene ligera ventaja, posiblemente debido a patrones no lineales en los datos.
3. La regresión logística ofrece ventajas en interpretabilidad y eficiencia computacional.
4. El desempeño de ambos modelos refuerza las conclusiones del Problema 2 sobre la importancia de evaluar múltiples enfoques según las características específicas del problema.

La exploración sistemática de hiperparámetros fue crucial, mejorando el F1 Score de 0.8205 (configuración base) a 0.8406 (configuración óptima), representando una mejora del 2.4%.

5 Problema 5

5.1 Enunciado

Considere el conjunto de datos de [3]. Cree un script con sklearn para resolver el problema de clasificación multiclase asociado con la clasificación de cervezas por estilo. Utilice el modelo KNN. Anote todas las suposiciones y operaciones de preprocesamiento de datos que realice. Investigue qué es una matriz de confusión. Luego, incorpórela a su análisis [4].

5.2 Metodología

Para resolver este problema de clasificación multiclase se siguió un proceso similar al Problema 3, con la variación de que ahora se trata de clasificar múltiples clases y se incorpora una nueva métrica de evaluación: la matriz de confusión. Las etapas del proceso fueron:

1. Carga del dataset 'beer.csv' desde Kaggle
2. Exploración de datos para entender su estructura y contenido
3. Preprocesamiento de datos
4. Entrenamiento del modelo KNN para clasificación multiclase
5. Evaluación del modelo utilizando métricas de rendimiento, incluyendo la matriz de confusión

Matriz de confusión: Antes de proceder con la implementación, es importante entender esta nueva métrica. La matriz de confusión es una herramienta que se utiliza para evaluar el rendimiento de un modelo de clasificación. Es una tabla que muestra las predicciones del modelo en comparación con las etiquetas reales. En clasificación binaria, se compone de cuatro elementos principales: Verdaderos Positivos (TP), Falsos Positivos (FP), Verdaderos Negativos (TN) y Falsos Negativos (FN). Estos elementos se utilizan para calcular métricas como la precisión, la sensibilidad y la especificidad del modelo. En el caso de un problema de clasificación multiclase, la matriz de confusión se extiende para incluir todas las clases posibles, lo que permite evaluar el rendimiento del modelo en cada clase individualmente.

Durante la fase de carga y exploración de datos, se identificó que el dataset cuenta con 150 observaciones y 6 columnas. La variable objetivo `style` representa el estilo de cerveza, con tres clases balanceadas:

- Premium Lager (50 muestras)
- IPA (50 muestras)
- Light Lager (50 muestras)

Las características disponibles son:

- Brew No: Número de lote de la cerveza
- OG: Gravedad original (cantidad de azúcar fermentable antes de la fermentación)
- BV: Alcohol por volumen
- pH: Acidez de la cerveza
- IBU: Indicador del amargor de la cerveza

De estas características, solo `Brew No.` no aporta información relevante al ser un valor consecutivo, por lo que fue eliminada. Las demás no presentan valores faltantes y están todas en formato numérico, por lo que no requieren transformaciones adicionales.

El preprocesamiento consistió en:

División de datos: Se separaron características y variable objetivo, eliminando `Brew No.`. Se realizó división train-test (80/20) con `stratify=y` para mantener el balance de clases, resultando en 120 muestras para entrenamiento y 30 para prueba.

Normalización: Se aplicó `StandardScaler` para normalizar las características, crucial para KNN por su sensibilidad a la escala.

Para el entrenamiento se utilizó la misma clase `KNNModelMulticlass` que extiende el modelo base para clasificación multiclase:

```
class KNNModelMulticlass:
    def __init__(self, n_neighbors=3):
        self.n_neighbors = n_neighbors
        self.model = KNeighborsClassifier(n_neighbors=self.
            n_neighbors)
        self.scaler = StandardScaler()

    def fit(self, X_train, y_train):
        X_train_scaled = self.scaler.fit_transform(X_train)
        self.model.fit(X_train_scaled, y_train)
        y_train_pred = self.model.predict(X_train_scaled)
        train_acc = self.model.score(X_train_scaled, y_train
            )
        train_f1 = f1_score(y_train, y_train_pred, average='
            weighted')
        return train_acc, train_f1

    def predict(self, X_test, y_test):
        X_test_scaled = self.scaler.transform(X_test)
        y_pred = self.model.predict(X_test_scaled)
        test_acc = self.model.score(X_test_scaled, y_test)
        test_f1 = f1_score(y_test, y_pred, average='weighted
            ')
        return y_pred, test_acc, test_f1
```

Se utilizó `average='weighted'` en el cálculo del F1 Score para considerar el peso de cada clase en problemas multiclase. Se probó inicialmente con $k = 5$ (configuración base) y posteriormente se exploraron valores de k desde 1 hasta 50 para encontrar el óptimo.

5.3 Resultados

El modelo KNN base con $k = 5$ obtuvo resultados excepcionales:

- **Train:** Accuracy = 1.0000, F1 Score = 1.0000
- **Test:** Accuracy = 1.0000, F1 Score = 1.0000

Este resultado perfecto sugiere que el dataset es relativamente sencillo de clasificar o podría indicar sobreajuste.

La búsqueda del valor óptimo de k reveló que a partir de $k = 2$ se obtiene un rendimiento perfecto (Accuracy y F1 Score de 1.0000). El único valor que obtuvo un rendimiento ligeramente inferior fue $k = 1$:

- $k = 1$**:** Test Accuracy = 0.9667, Test F1 Score = 0.9667
- $k \geq 2$**:** Test Accuracy = 1.0000, Test F1 Score = 1.0000

La Figura 2 muestra la evolución del desempeño del modelo. Se observa que $k = 1$ tiene un pequeño error, pero a partir de $k = 2$ el modelo alcanza perfección total y se mantiene constante.

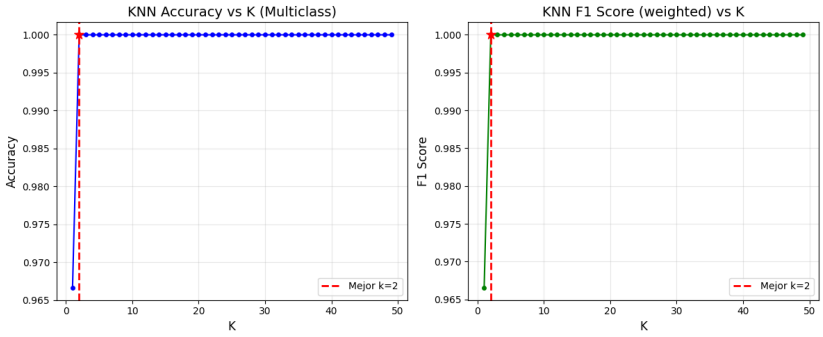

Figure 2: Evolución del Accuracy y F1 Score en función del hiperparámetro k para el modelo KNN multiclase. Se observa rendimiento perfecto desde $k = 2$ en adelante.

Dado que analizar la matriz de confusión es la parte más importante de este problema, se compararon las matrices para $k = 1$ y $k = 2$. La Figura 3 muestra que para $k = 1$, el modelo cometió un error de clasificación: una muestra de la clase "Light Lager" fue incorrectamente clasificada como "Premium Lager". Esto se refleja en la matriz de confusión donde Light Lager tiene 9 predicciones correctas en lugar de 10.

Por otro lado, la Figura 4 muestra la matriz de confusión para $k = 2$, donde todas las muestras fueron clasificadas correctamente. La diagonal principal contiene todos los valores predichos correctamente (10 para cada clase), sin errores de clasificación fuera de la diagonal.

5.4 Discusión

Los resultados excepcionales obtenidos revelan varios aspectos importantes:

Rendimiento perfecto desde $k = 2$**:** Las características del dataset (OG, ABV, pH, IBU) son altamente discriminativas, permitiendo que las tres clases estén bien separadas en el espacio de características. Solo 2 vecinos cercanos son suficientes para clasificación perfecta.

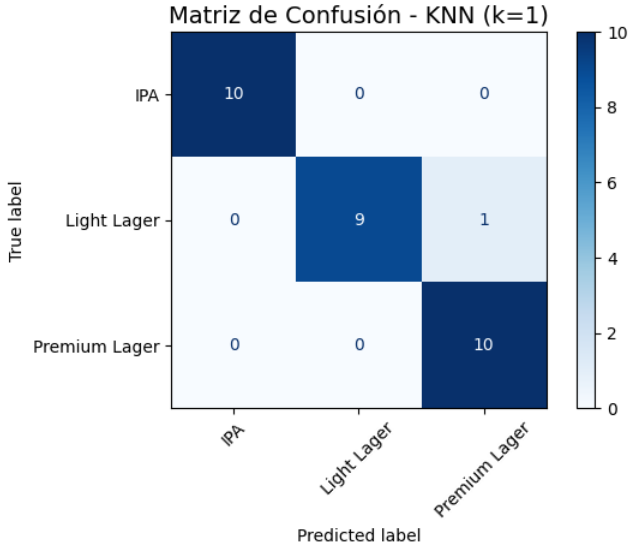

Figure 3: Matriz de confusión para KNN con $k = 1$. Se observa un error en la clasificación de Light Lager, confundida con Premium Lager.

Error de $k = 1$: La única muestra mal clasificada fue una Light Lager confundida con Premium Lager, indicando un punto cercano a la frontera entre estas clases. La votación con $k = 2$ corrige este error.

Matriz de confusión: El error entre Light Lager y Premium Lager es comprensible, ya que ambos lagers comparten características similares (bajo IBU y ABV). La IPA, al ser más distintiva, no presentó errores.

Dataset sencillo vs sobreajuste: El rendimiento perfecto con múltiples valores de k (2 a 50) sugiere que el dataset es naturalmente separable, no sobreajuste. Un modelo sobreajustado mostraría degradación con k mayores.

Balance de clases: Las 50 muestras por clase facilitaron el entrenamiento sin sesgos, mantenido mediante `stratify` en la división train-test.

5.5 Conclusión

El modelo KNN demostró ser extremadamente efectivo para clasificar los estilos de cerveza en este dataset, alcanzando un rendimiento perfecto (Accuracy y F1 Score de 1.0000) con $k \geq 2$.

La matriz de confusión demostró ser una herramienta fundamental para entender el comportamiento del modelo en cada clase individual. Permitió identificar que el único error de clasificación con $k = 1$ ocurrió entre Light Lager y Premium Lager, dos estilos que comparten características similares al ser ambos lagers.

Puntos clave del análisis:

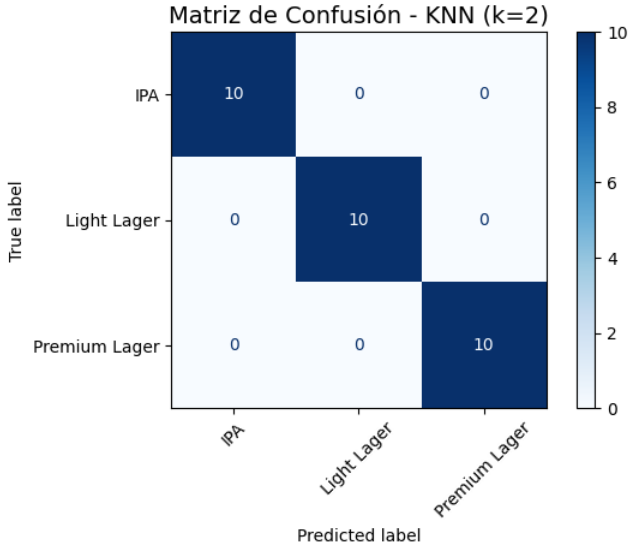

Figure 4: Matriz de confusión para KNN con $k = 2$ (valor óptimo). Todas las muestras fueron clasificadas correctamente, resultando en una matriz de confusión perfecta.

1. El valor óptimo encontrado fue $k = 2$, que proporciona clasificación perfecta con el mínimo número de vecinos necesario.

2. Las características del dataset (OG, ABV, pH, IBU) son altamente discriminativas para los tres estilos de cerveza.

3. La matriz de confusión reveló que IPA es la clase más fácilmente distinguible, sin errores en ninguna configuración.

4. El error único con $k = 1$ entre Light Lager y Premium Lager sugiere que estos estilos tienen mayor similitud en el espacio de características.

5. El rendimiento perfecto con múltiples valores de k sugiere que el dataset es naturalmente separable más que indicativo de sobreajuste.

Es importante mencionar que, aunque el resultado perfecto es alentador, sería recomendable realizar validación cruzada para confirmar la robustez del modelo en diferentes particiones de los datos. Adicionalmente, podría considerarse usar una proporción menor para entrenamiento (ej. 70/30 o 60/40) para evaluar si el modelo mantiene su capacidad predictiva con menos datos de entrenamiento.

6 Problema 6

6.1 Enunciado

Busque los términos "One Vs One Classifie" y "One Vs Rest Classifie". Luego, repita el problema 5 usando sklearn para los dos enfoques mencionados. Anote todas las suposiciones y conclusiones.

6.2 Metodología

Para este problema se busca comparar distintos enfoques para la clasificación multiclase utilizando el mismo dataset de cervezas del Problema 5. Los enfoques a comparar son "One-vs-One Classifier" y "One-vs-Rest Classifier". Es importante entender qué son estos enfoques:

One-vs-One Classifier: Este enfoque consiste en entrenar un modelo de clasificación binaria para cada par de clases en el dataset. En este caso, dado que tenemos 3 clases (Premium Lager, IPA, Light Lager), este enfoque entrenaría 3 modelos de clasificación binaria:

- Premium Lager vs IPA
- Premium Lager vs Light Lager
- IPA vs Light Lager

Para hacer una predicción, se utiliza un sistema de votación donde cada modelo de clasificación binaria emite un voto por la clase que predice, y la clase con más votos es la predicción final.

One-vs-Rest Classifier: Este enfoque consiste en entrenar un modelo de clasificación binaria para cada clase en el dataset, donde cada modelo se entrena para distinguir una clase específica de todas las demás clases combinadas. En este caso, dado que tenemos 3 clases, este enfoque entrenaría 3 modelos de clasificación binaria:

- Premium Lager vs (IPA + Light Lager)
- IPA vs (Premium Lager + Light Lager)
- Light Lager vs (Premium Lager + IPA)

Para hacer una predicción, se utiliza un sistema de votación donde cada modelo de clasificación binaria emite un voto por la clase que predice, y la clase con más votos es la predicción final.

Para la implementación de estos enfoques utilizando sklearn, se puede utilizar la clase **OneVsOneClassifier** para el enfoque One-vs-One y la clase **OneVsRestClassifier** para el enfoque One-vs-Rest. Ambos enfoques se pueden implementar utilizando el modelo KNN como clasificador base. La manera en la que esto funciona es inicializando la clase **OneVsOneClassifier** o **OneVsRestClassifier** y pasando como argumento el modelo KNN, luego se ajusta el

modelo con los datos de entrenamiento y se realizan las predicciones con los datos de prueba.

```
from sklearn.multiclass import OneVsOneClassifier,
    OneVsRestClassifier

# One-vs-One
ovo_model = OneVsOneClassifier(KNeighborsClassifier(
    n_neighbors=k))

# One-vs-Rest
ovr_model = OneVsRestClassifier(KNeighborsClassifier(
    n_neighbors=k))
```

Para la parte de implementación, se siguió el mismo enfoque que en el Problema 5, simplemente omitiendo la parte de exploración, ya que esta se realizó en el problema anterior, y se procedió directamente al preprocesamiento de datos, entrenamiento del modelo y evaluación del rendimiento utilizando las mismas métricas que en el Problema 5, incluyendo la matriz de confusión.

Otro cambio en la implementación es la modificación a la clase que se tenía para entrenar/probar KNN multiclase, añadiendo como parámetro el tipo de enfoque a utilizar, siendo este `'ovo'` para One-vs-One y `'ovr'` para One-vs-Rest.

Para evitar los problemas de sobreajuste que se presentaron en el Problema 5, se optó por realizar una división del dataset con un 10% para entrenamiento y un 90% para prueba, con el fin de tener un modelo que generalice mejor a datos no vistos y no obtener resultados perfectos que puedan ser un indicativo de sobreajuste.

6.3 Resultados

Los resultados obtenidos muestran que los 3 enfoques (base vs ovo vs ovr) obtuvieron un rendimiento prácticamente igual:

Estrategia	**Train Acc**	**Train F1**	**Test Acc**	**Test F1**
standard	0.8667	0.8611	0.9259	0.9250
ovr	0.8667	0.8611	0.9259	0.9250
ovo	0.8667	0.8611	0.9259	0.9250

Al analizar las matrices de confusión para los 3 enfoques (Figura 5), se aprecia que todos cometieron el mismo error de clasificación, clasificando incorrectamente 10 muestras de la clase Premium Lager como Light Lager, lo que se refleja en una matriz de confusión con un único error de clasificación para ambas estrategias.

6.4 Discusión

Equivalencia de estrategias: Los resultados idénticos entre las tres estrategias (estándar, OvO, OvR) se pueden explicar por el hecho de que el dataset

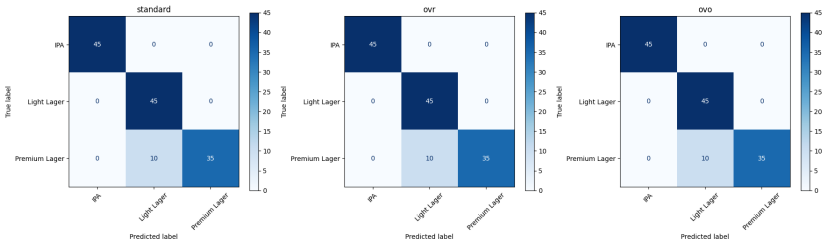

Figure 5: Matrices de confusión para los tres enfoques (Standard, One-vs-Rest, One-vs-One) con $k = 5$. Todos muestran el mismo patrón: 10 muestras de Premium Lager mal clasificadas como Light Lager.

es relativamente sencillo de clasificar. El modelo KNN es capaz de clasificar correctamente las muestras de prueba con cualquier enfoque, ya que las clases están relativamente bien separadas en el espacio de características.

Impacto de la división train-test: La estrategia de aumentar drásticamente el tamaño del conjunto de prueba (10/90 en lugar de 80/20) se hizo con el fin de evitar los problemas de sobreajuste que se presentaron en el Problema 5. Esta modificación permitió obtener resultados más realistas (Accuracy de 0.9259 vs 1.0) que reflejan mejor la capacidad del modelo para generalizar a datos no vistos con solo 5 muestras de entrenamiento por clase.

Patrón de errores consistente: Los 10 errores de clasificación ocurrieron consistentemente entre Premium Lager y Light Lager en todos los enfoques. Esto refuerza la observación del Problema 5 sobre la similitud entre estos dos estilos de lager, mientras que la IPA se distingue fácilmente por sus características más distintivas (mayor IBU y ABV).

Robustez del modelo: El hecho de que el modelo mantenga un F1 Score de 0.9250 con solo 15 muestras de entrenamiento demuestra que las características del dataset (OG, ABV, pH, IBU) son lo suficientemente informativas para permitir buena clasificación incluso con muy pocos ejemplos por clase.

6.5 Conclusión

Se puede concluir que tanto el enfoque One-vs-One como el enfoque One-vs-Rest son igualmente efectivos para este problema de clasificación multiclase utilizando el modelo KNN, alcanzando un Accuracy de 0.9259 y F1 Score de 0.9250.

Puntos clave:

1. Los tres enfoques (estándar, OvO, OvR) producen resultados idénticos, tanto en métricas como en patrones de error.

2. La división 10/90 proporcionó una evaluación más realista de la generalización, evitando los resultados perfectos del Problema 5.

3. Los errores ocurrieron consistentemente entre Premium Lager y Light Lager, confirmando la similitud entre estos estilos.

4. El modelo KNN demostró robustez significativa, manteniendo desempeño superior al 92% con solo 5 muestras de entrenamiento por clase.

La equivalencia de resultados sugiere que para este dataset específico, la elección de la estrategia multiclase no es crítica, pudiendo basarse la selección en consideraciones de eficiencia computacional o interpretabilidad.

7 Problema 7

7.1 Enunciado

Considere la ecuación cuadrática general.

$$ax^2 + bx + c = 0$$

Esfuércese por clasificar las raíces de la ecuación anterior en función de sus coeficientes. Para simplificar, debe fijar el dominio de x y proponer un dominio adecuado para los coeficientes. Escriba todas sus suposiciones y muestre sus hallazgos mediante gráficos.

7.2 Metodología

Para dar solución a este problema es necesario aplicar todo lo que se estuvo viendo a lo largo de la resolución de todos los problemas anteriores, con el añadido de que esta vez somos nosotros quienes tenemos que proponer la forma adecuada para dar solución a este problema mediante un conjunto de datos sintéticos, ya que no contamos con un dataset como tal.

Suposiciones:

Para abordar este problema se establecieron las siguientes suposiciones:

- El dominio de x es el conjunto de los números reales $\mathbb{R}$.
- El dominio de los coeficientes a, b y c es el conjunto de los números reales $\mathbb{R}$, con la restricción de que $a \neq 0$, ya que de lo contrario no sería una ecuación cuadrática al eliminar el término cuadrático por el cual está multiplicando.
- Para clasificar las raíces de la ecuación cuadrática, se utilizará el discriminante, que se define como $\Delta = b^2 - 4ac$. La clasificación de las raíces se realiza de la siguiente manera:
  - Si $\Delta > 0$, la ecuación tiene dos raíces reales y distintas.
  - Si $\Delta \approx 0$, la ecuación tiene una raíz real doble (o dos raíces reales iguales).
  - Si $\Delta < 0$, la ecuación no tiene raíces reales, sino que tiene dos raíces complejas conjugadas.

- A los coeficientes a, b y c se les asignarán valores aleatorios dentro de un rango específico para generar un conjunto de datos sintéticos. Para este caso se eligió el rango de -10 a 10 para cada uno de los coeficientes (asegurándonos de que $a \neq 0$).

Selección del modelo:

Dado que este es un problema de clasificación, a lo largo de este trabajo hemos visto 2 enfoques principales para abordar este tipo de problemas, siendo estos la regresión logística y KNN. Como se vio en el Problema 2 donde se hacía el análisis comparativo de ambos modelos, se concluyó que para problemas no lineales, el modelo de KNN es el que mejor se desempeña, y como en este caso nos enfrentamos a un problema cuadrático, lo ideal es utilizar KNN multiclase, donde nuestras clases serán:

- **Clase 0:** Raíces complejas conjugadas ($\Delta < 0$)
- **Clase 1:** Raíz real doble ($\Delta \approx 0$)
- **Clase 2:** Raíces reales y distintas ($\Delta > 0$)

Generación del dataset sintético:

Ya habiendo definido las reglas para este problema, se implementó una función que genera un conjunto de datos sintéticos con los coeficientes a, b y c, calculando el discriminante para clasificar cada conjunto de coeficientes en una de las tres clases mencionadas. La función garantiza un balance perfecto generando exactamente 1000 ecuaciones por clase:

```
def generate_quadratic_data(n_samples=3000):
    data = []
    samples_per_class = n_samples // 3

    # Clase 2: Raices reales distintas (Delta > 0)
    count = 0
    while count < samples_per_class:
        a = np.random.uniform(-10, 10)
        while abs(a) < 0.1:
            a = np.random.uniform(-10, 10)
        b = np.random.uniform(-10, 10)
        max_c = (b**2) / (4 * abs(a)) - 0.1
        c = np.random.uniform(-10, min(10, max_c))
        discriminant = b**2 - 4*a*c
        if discriminant > 0:
            data.append([a, b, c, 2])
            count += 1

    # Clase 1: Raiz real doble (Delta = 0)
    count = 0
    while count < samples_per_class:
        a = np.random.uniform(-10, 10)
```

```
        while abs(a) < 0.1:
            a = np.random.uniform(-10, 10)
        b = np.random.uniform(-10, 10)
        c = (b**2) / (4 * a) + np.random.uniform(-0.05,
            0.05)
        data.append([a, b, c, 1])
        count += 1

    # Clase 0: Raices complejas (Delta < 0)
    count = 0
    while count < samples_per_class:
        a = np.random.uniform(-10, 10)
        while abs(a) < 0.1:
            a = np.random.uniform(-10, 10)
        b = np.random.uniform(-10, 10)
        min_c = (b**2) / (4 * abs(a)) + 0.1
        c = np.random.uniform(min(min_c, 10), 10)
        discriminant = b**2 - 4*a*c
        if discriminant < 0:
            data.append([a, b, c, 0])
            count += 1

    df = pd.DataFrame(data, columns=['a', 'b', 'c', '
        root_type'])
    return df
```

Esta función genera 3000 ecuaciones cuadráticas (1000 por clase), asegurando que $|a| \geq 0.1$ para evitar ecuaciones degeneradas. Para la Clase 1, se añade una pequeña perturbación (± 0.05) alrededor de $c = b^2/(4a)$ para simular condiciones cercanas al discriminante cero.

Preprocesamiento de datos:

A estos datos solamente se les realizó un preproceso de división en conjuntos de entrenamiento (80%) y prueba (20%), y luego se normalizaron utilizando `StandardScaler` para mejorar el rendimiento del modelo KNN. No fue necesario realizar imputación de valores faltantes ni codificación de variables categóricas, ya que todas las características son numéricas y el dataset se generó sin valores faltantes.

Entrenamiento del modelo:

Finalmente, se entrenó el modelo KNN con los datos de entrenamiento utilizando la clase **KNNQuadraticClassifier** que encapsula la funcionalidad del modelo:

```
class KNNQuadraticClassifier:
    def __init__(self, n_neighbors=5):
        self.n_neighbors = n_neighbors
        self.model = KNeighborsClassifier(n_neighbors=
            n_neighbors)
        self.scaler = StandardScaler()
```

```
    def fit(self, X_train, y_train):
        X_train_scaled = self.scaler.fit_transform(X_train)
        self.model.fit(X_train_scaled, y_train)
        y_train_pred = self.model.predict(X_train_scaled)
        train_acc = accuracy_score(y_train, y_train_pred)
        train_f1 = f1_score(y_train, y_train_pred, average='
            weighted')
        return train_acc, train_f1

    def predict(self, X_test, y_test):
        X_test_scaled = self.scaler.transform(X_test)
        y_pred = self.model.predict(X_test_scaled)
        test_acc = accuracy_score(y_test, y_pred)
        test_f1 = f1_score(y_test, y_pred, average='weighted
            ')
        return y_pred, test_acc, test_f1
```

Se evaluó su desempeño utilizando los datos de prueba, obteniendo métricas como el accuracy, F1 score y matriz de confusión como se ha visto en los problemas anteriores. Se realizó una búsqueda exhaustiva del mejor valor de k probando desde 1 hasta 100.

7.3 Resultados

La búsqueda del valor óptimo de k reveló que $k = 1$ es el mejor valor para este dataset:

El modelo final con $k = 1$ obtuvo los siguientes resultados:

- **Train:** Accuracy = 1.0000, F1 Score = 1.0000
- **Test:** Accuracy = 0.8950, F1 Score = 0.8954

La Figura 6 muestra la evolución del F1 Score en función del valor de k. Se observa que $k = 1$ proporciona el mejor rendimiento, y conforme aumenta el valor de k, el rendimiento del modelo disminuye gradualmente. Esto sugiere que considerar solo el vecino más cercano es suficiente para este problema, y agregar más vecinos introduce ruido que perjudica la clasificación.

La matriz de confusión (Figura 7) revela el patrón de clasificación del modelo. Se aprecia que la mayoría de los aciertos se concentran en la diagonal principal, con cada clase teniendo más de 170 muestras correctamente clasificadas de un total de 200 muestras por clase (600 en total para prueba).

Del análisis de la matriz de confusión se observa que:

- **Clase 0 (Raíces complejas):** 183 correctas, 17 confundidas con Clase 1, 0 confundidas con Clase 2.
- **Clase 1 (Raíz real doble):** 172 correctas, 19 confundidas con Clase 0, 9 confundidas con Clase 2.

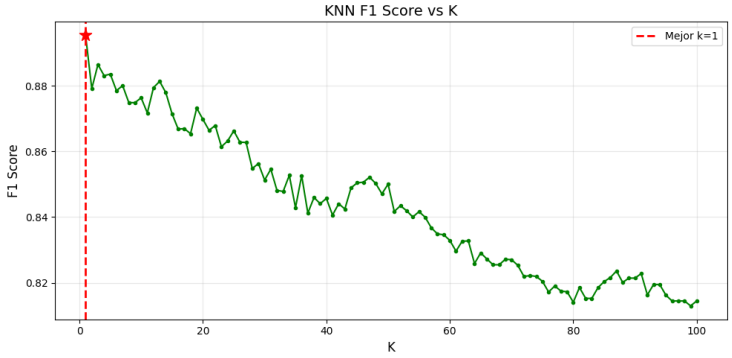

Figure 6: Evolución del F1 Score en función del hiperparámetro k para el modelo KNN en clasificación de raíces cuadráticas. Se observa que $k = 1$ proporciona el mejor rendimiento, con degradación gradual al aumentar k.

- **Clase 2 (Raíces reales distintas):** 182 correctas, 2 confundidas con Clase 0, 16 confundidas con Clase 1.

La Clase 1 (raíz real doble) es la que presenta una mayor cantidad de errores, lo cual se puede deber a que esta clase puede ser más difícil de clasificar debido a que las raíces reales dobles se encuentran en la frontera entre las raíces complejas y las raíces reales distintas, lo cual puede llevar a una mayor confusión con las otras clases.

7.4 Discusión

Los resultados obtenidos revelan varios aspectos importantes sobre el problema de clasificación de raíces cuadráticas:

Rendimiento óptimo con $k = 1$: El hecho de que el mejor rendimiento se obtenga con $k = 1$ indica que el espacio de características está bien estructurado, donde cada ecuación cuadrática está más cerca de otras ecuaciones de su misma clase. Esto sugiere que los coeficientes a, b y c proporcionan suficiente información discriminativa cuando se consideran conjuntamente. El modelo logró bastarse únicamente con un $k = 1$ para obtener un resultado bastante bueno, lo cual se puede deber a que el dataset es bastante sencillo y no presenta mucha complejidad.

Degradación con k mayor: Al aumentar el valor de k, el modelo comienza a tener un rendimiento peor, lo cual se puede deber a que al aumentar el valor de k, el modelo comienza a considerar más vecinos para tomar la decisión de clasificación, lo cual puede llevar a una mayor confusión entre las clases, especialmente en las zonas de frontera donde el discriminante está cerca de cero.

Rendimiento general: Un F1 Score de 0.8954 (89.54%) es un resultado bastante bueno considerando que estamos clasificando un fenómeno matemático con fronteras naturalmente difusas, especialmente en la región donde $\Delta \approx 0$. El

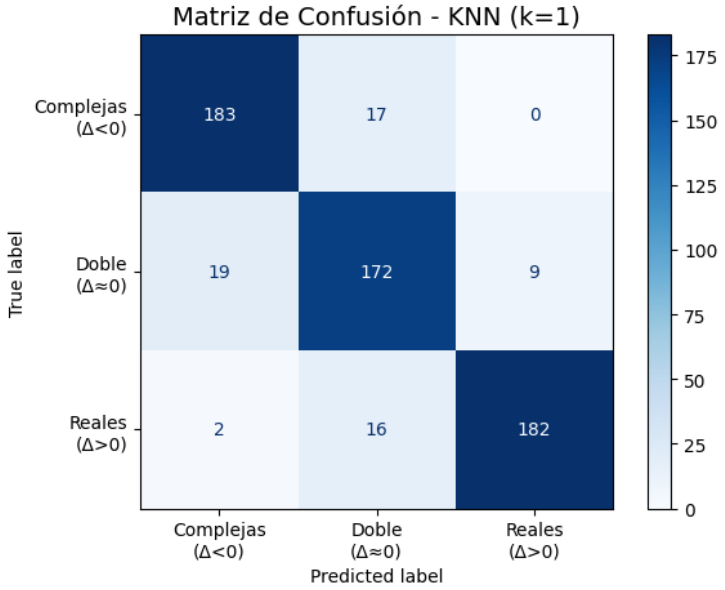

Figure 7: Matriz de confusión para KNN con $k = 1$. Las tres clases muestran buen desempeño, aunque la Clase 1 (raíz real doble) presenta mayor cantidad de errores, principalmente confundida con la Clase 2 (raíces reales distintas).

rendimiento perfecto en entrenamiento (1.0000) combinado con 0.8954 en prueba sugiere que el modelo ha aprendido bien los patrones sin sobreajuste excesivo.

Variabilidad del dataset: Es importante tener en cuenta que los resultados pueden variar dependiendo de la generación del dataset sintético y de la distribución de los coeficientes, por lo que se recomienda realizar múltiples ejecuciones con diferentes conjuntos de datos para obtener una evaluación más robusta del modelo.

7.5 Conclusión

En conclusión, el modelo KNN demostró ser efectivo para clasificar las raíces de la ecuación cuadrática en función de sus coeficientes, alcanzando un F1 Score de 0.8954 y Accuracy de 0.8950 en el conjunto de prueba.

Puntos clave del análisis:

1. La elección de $k = 1$ resultó ser la mejor opción para este conjunto de datos sintético, indicando que el vecino más cercano proporciona la información más relevante para la clasificación.

2. El enfoque basado en los coeficientes (a, b, c) como características demostró ser efectivo para predecir el tipo de raíces, validando que el discriminante

$\Delta = b^2 - 4ac$ captura implícitamente información relevante a través de estos coeficientes.

3. La Clase 1 (raíz real doble, $\Delta \approx 0$) presentó mayor dificultad de clasificación, lo cual es matemáticamente consistente al representar una frontera continua entre dos regiones discretas.

4. La matriz de confusión reveló que los errores están distribuidos de manera relativamente uniforme entre las clases, sin sesgos significativos hacia ninguna clase en particular.

5. El modelo alcanzó un rendimiento robusto del 89.54% en un problema que involucra fronteras matemáticas naturalmente difusas, demostrando la viabilidad del enfoque de machine learning para problemas de clasificación matemática.

References

[1] `https://www.kaggle.com/datasets/yasserh/horsesurvivalprognostication`.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`.

[3] `https://www.kaggle.com/datasets/ankurnapa/ankurs-beer-data-set`.

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html`.

[5] G. James et al. *An Introduction to Statistical Learning - with Applications in Python*. Springer, 2023.