Práctica 3

Competición Kaggle: Clasificación de Hojas de Tomate

Inteligencia de Negocio

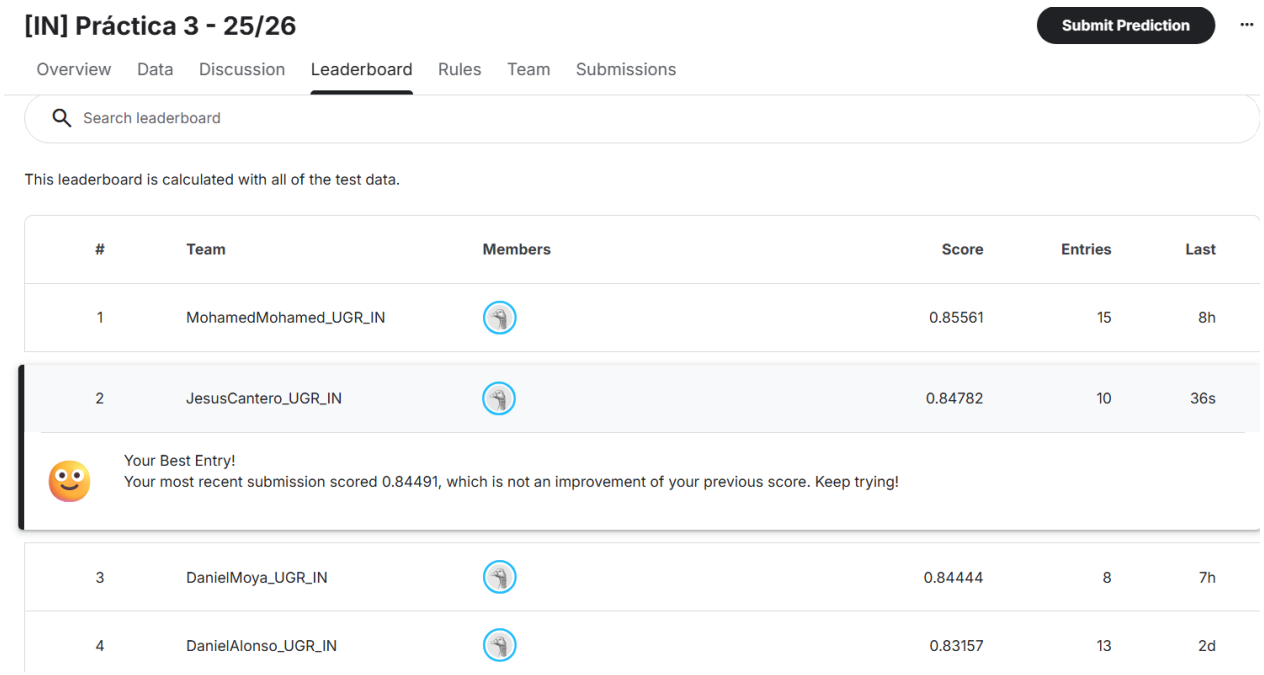
Jesús J. Cantero

Grupo A

jesusjcl@correo.ugr.es

Curso 2025–2026

Grado en Ingeniería Informática
Universidad de Granada (Ceuta)



Captura del Leaderboard de Kaggle – Fecha: 01/01/2026

Resumen de participación:

Usuario:	JesusCantero_UGR_IN
Posición actual:	2 de 5
Mejor Score:	0.84782
Número de entries:	10

Índice

1. Introducción

1.1. Contexto y Motivación

Esta práctica consiste en una competición Kaggle de clasificación binaria de hojas de tomate. El objetivo es distinguir entre hojas sanas (*control*) y hojas infectadas por el hongo *Botrytis cinerea* (*botrytis*) utilizando datos de fluorescencia multicolor e imágenes hiperespectrales.

La detección temprana de enfermedades en cultivos es fundamental para la agricultura de precisión y la gestión eficiente de recursos. Las técnicas de aprendizaje automático permiten automatizar el diagnóstico a partir de mediciones no destructivas, reduciendo pérdidas y optimizando el uso de tratamientos fitosanitarios.

1.2. Descripción del Dataset

El dataset proporcionado contiene mediciones de hojas de tomate:

- **Conjunto de entrenamiento**: 336 muestras con etiquetas
- **Conjunto de test**: 143 muestras sin etiquetas
- **Total de variables**: 309 columnas

Tipo	Columnas	Descripción
Metadatos (NO USAR)	`exp`, `dpi`, `leaf`, `spot`	Información experimental
Fluorescencia	`F440`, `F520`, `F680`, `F740`	4 valores de fluorescencia
Hiperespectral	`w388.13` a `w1028.28`	300 variables espectrales
Target	`class`	`control` (0) o `botrytis` (1)

Cuadro 1: Descripción de las variables del dataset.

1.3. Herramientas y Tecnologías

- **pandas** y **numpy**: manipulación de datos
- **scikit-learn**: modelos de clasificación, escalado y validación cruzada
- **XGBoost** y **LightGBM**: algoritmos de gradient boosting
- **matplotlib** y **seaborn**: visualización
- **Jupyter Notebook**: análisis exploratorio interactivo

1.4. Métrica de Evaluación

La métrica utilizada en la competición es el **F1-score**, definido como:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Esta métrica es apropiada para problemas de clasificación binaria con posible desbalance de clases, ya que considera tanto los falsos positivos como los falsos negativos.

2. Análisis Exploratorio de Datos (EDA)

2.1. Distribución de Clases

El análisis de la distribución de clases revela un desbalance moderado:

Clase	Muestras	Porcentaje
botrytis	196	58.3 %
control	140	41.7 %

Cuadro 2: Distribución de clases en el conjunto de entrenamiento.

El ratio de desbalance es de 1.40:1, lo cual es moderado y no requiere técnicas agresivas de balanceo.

2.2. Variables de Fluorescencia

Las 4 variables de fluorescencia presentan las siguientes características:

Variable	Media	Std	Min	Max
F440	235.57	9.19	209.95	257.33
F520	431.88	31.66	327.26	517.98
F680	773.10	120.98	484.27	1270.81
F740	1403.55	227.24	795.12	2125.33

Cuadro 3: Estadísticas descriptivas de las variables de fluorescencia.

Se observan dos grupos de variables correlacionadas:

- **Grupo 1**: F440 y F520 (correlación positiva: 0.89)
- **Grupo 2**: F680 y F740 (correlación positiva: 0.96)

Existe correlación negativa entre ambos grupos, sugiriendo características complementarias.

2.3. Variables Espectrales

El dataset incluye 300 variables espectrales en el rango de 388.13 nm a 1028.28 nm. El análisis de espectros promedio por clase muestra diferencias sutiles pero consistentes entre hojas sanas e infectadas.

2.4. Análisis de Valores Faltantes y Outliers

- **Valores faltantes**: No se detectaron valores faltantes en ninguno de los conjuntos.
- **Outliers** (método IQR en fluorescencia):
  - F440: 4 outliers (1.2 %)
  - F520: 10 outliers (3.0 %)
  - F680: 6 outliers (1.8 %)
  - F740: 4 outliers (1.2 %)

2.5. Reducción de Dimensionalidad (PCA)

El análisis de componentes principales revela alta reducibilidad dimensional:

- **3 componentes** explican el 95 % de la varianza
- **7 componentes** explican el 99 % de la varianza

Esto indica que, a pesar de tener 304 features, la información relevante se concentra en pocas dimensiones.

3. Preprocesamiento de Datos

3.1. Limpieza de Datos

Durante la carga de datos se detectó un problema de formato: algunos valores numéricos contenían espacios (ej: `'232 .25'`). Se implementó una función de limpieza automática:

```
def clean_numeric_columns(df):
    for col in df.columns:
        if df[col].dtype == 'object' and col not in METADATA_COLS + [TARGET_COL]:
            df[col] = df[col].astype(str).str.replace('␣', '').astype(float)
    return df
```

3.2. Preparación de Features

El pipeline de preprocesamiento incluye:

1. **Exclusión de metadatos**: Se eliminan las columnas `exp`, `dpi`, `leaf`, `spot`
2. **Codificación del target**: `control` $\rightarrow$ 0, `botrytis` $\rightarrow$ 1
3. **Escalado**: StandardScaler para normalizar las features

3.3. Arquitectura del Código

El código se organiza en módulos reutilizables:

- `src/preprocessing.py`: Funciones de carga, limpieza y transformación
- `src/models.py`: Definición y evaluación de modelos
- `src/utils.py`: Utilidades para submissions y logging

4. Modelado y Experimentación

4.1. Metodología

Se utilizó validación cruzada estratificada con 5 folds para evaluar los modelos, asegurando que cada fold mantiene la proporción de clases del dataset original.

4.2. Experimento 01: Modelos Baseline

Se compararon 5 algoritmos de clasificación con parámetros por defecto:

Modelo	F1-Score (CV)	Desv. Std
Logistic Regression	**0.8278**	0.0232
SVM (RBF)	0.7624	0.0397
Random Forest	0.7530	0.0510
Gradient Boosting	0.7371	0.0314
KNN (k=5)	0.6558	0.0681

Cuadro 4: Comparación de modelos baseline con StandardScaler.

Resultado: Logistic Regression obtuvo el mejor F1-score en validación cruzada (0.8278), siendo seleccionado para la primera submission.

4.3. Configuración del Mejor Modelo

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Preprocesamiento
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Modelo
model = LogisticRegression(max_iter=1000, random_state=42)
model.fit(X_train_scaled, y_train)

# Predicciones
predictions = model.predict(X_test_scaled)
```

5. Registro de Experimentos

La siguiente tabla contiene el registro obligatorio de todas las submissions realizadas a Kaggle, incluyendo fecha/hora, posición en el momento de la subida, scores de entrenamiento y test, preprocesado aplicado, algoritmo utilizado y configuración de parámetros.

Nº	Fecha	Pos.	F1 CV	F1 Kaggle	Preprocesado	Algoritmo	Descripción
–	21/12/25	–	–	0.00000	–	–	Prueba (sample)
01a	23/12/25	1	0.8278	**0.84782**	Scaler	LR	Baseline (seed=42)
01b	23/12/25	1	∼0.83	**0.84782**	Scaler	LR	Baseline (otra seed)
02	25/12/25	–	0.7368	0.74561	Scaler+PCA(3)	LR	95 % varianza
03	25/12/25	–	0.7268	0.69364	Scaler+PCA(7)	XGBoost	99 % varianza
04	25/12/25	–	0.6802	0.59459	Scaler+SKB(50)	RF	ANOVA F-value
05	25/12/25	–	0.7472	0.70454	Scaler	XGBoost	est=100, d=6
06	25/12/25	–	0.7232	0.73033	Scaler	LightGBM	est=100, d=6
15	01/01/26	2	0.8518	0.82485	Scaler	Ridge	α=1.0
19	01/01/26	2	0.8517	0.84491	Spec+RFE(100)	LR	100 features

Cuadro 5: Registro completo de 10 submissions a Kaggle. LR=LogisticRegression, RF=RandomForest, Scaler=StandardScaler, SKB=SelectKBest, est=n_estimators, d=max_depth, Spec=Spectral.

5.1. Detalle del Experimento 01

- **Fecha y hora de subida**: 23/12/2025 20:30
- **Posición en el momento**: 1 de 2
- **Score en entrenamiento (CV 5-fold)**: 0.8278 ($\pm$ 0.0232)
- **Score en Kaggle (test)**: 0.84782
- **Preprocesado**:
  - Limpieza de valores numéricos con espacios
  - Exclusión de columnas de metadatos (exp, dpi, leaf, spot)
  - Normalización con StandardScaler
  - Sin reducción de dimensionalidad (304 features)
- **Algoritmo**: Logistic Regression
- **Parámetros**: `max_iter=1000, random_state=42`
- **Observaciones**: Modelo baseline. El score en Kaggle (0.84782) supera ligeramente al score de validación cruzada (0.8278), indicando buena generalización.

5.2. Fase 1: Reducción de Dimensionalidad

El objetivo de esta fase fue evaluar el impacto de diferentes técnicas de reducción de dimensionalidad en el rendimiento del modelo. Se implementaron tres enfoques principales:

5.2.1. Experimento 02: PCA(3) + Logistic Regression

Se aplicó Análisis de Componentes Principales reduciendo a 3 componentes, que capturan el 95 % de la varianza total. Los resultados mostraron una disminución significativa en el rendimiento:

- **F1-score CV**: 0.7368 (± 0.0020)
- **Observación**: Aunque se retiene la mayor parte de la varianza, se pierde información discriminativa crucial para la clasificación.

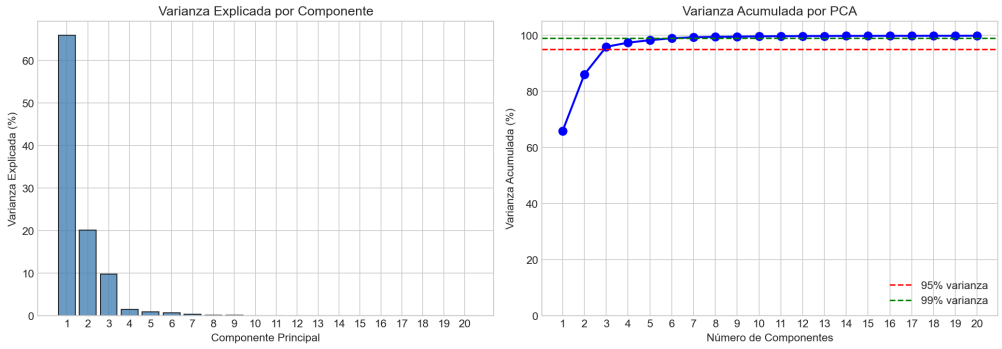


Figura 1: Varianza explicada por componentes principales.

5.2.2. Experimento 03: PCA(7) + XGBoost

Se evaluó PCA con 7 componentes (99 % de varianza) combinado con XGBoost:

- **F1-score CV**: 0.7268 (± 0.0508)
- **Observación**: Aumentar a 7 componentes no mejora significativamente respecto a PCA(3), y la alta varianza indica inestabilidad.

5.2.3. Experimento 04: SelectKBest(50) + Random Forest

Se seleccionaron las 50 mejores features mediante prueba F de ANOVA:

- **F1-score CV**: 0.6802 (± 0.0544)
- **Observación**: Selección agresiva de features elimina información importante, resultando en el peor rendimiento.

5.2.4. Conclusiones Fase 1

1. **La reducción de dimensionalidad no beneficia el rendimiento** en este problema específico.
2. **PCA(3) es la mejor opción** si se requiere reducción, pero aún inferior al baseline.
3. **SelectKBest es demasiado agresivo** con k=50 para este dataset.
4. **Se recomienda mantener todas las features** para los modelos avanzados.

5.3. Fase 2: Modelos Avanzados

En esta fase se implementaron algoritmos de boosting state-of-the-art para comparar con el baseline de Logistic Regression.

5.3.1. Experimento 05: XGBoost Baseline

XGBoost con parámetros por defecto sobre todas las features:

- **F1-score CV**: 0.7472 (± 0.0520)
- **Parámetros**: n_estimators=100, max_depth=6, learning_rate=0.1
- **Observación**: No supera al baseline, alta varianza indica necesidad de optimización.

5.3.2. Experimento 06: LightGBM Baseline

LightGBM como alternativa eficiente a XGBoost:

- **F1-score CV**: 0.7232 (± 0.0556)
- **Parámetros**: n_estimators=100, max_depth=6, learning_rate=0.1
- **Observación**: Rendimiento similar a XGBoost, también requiere optimización.

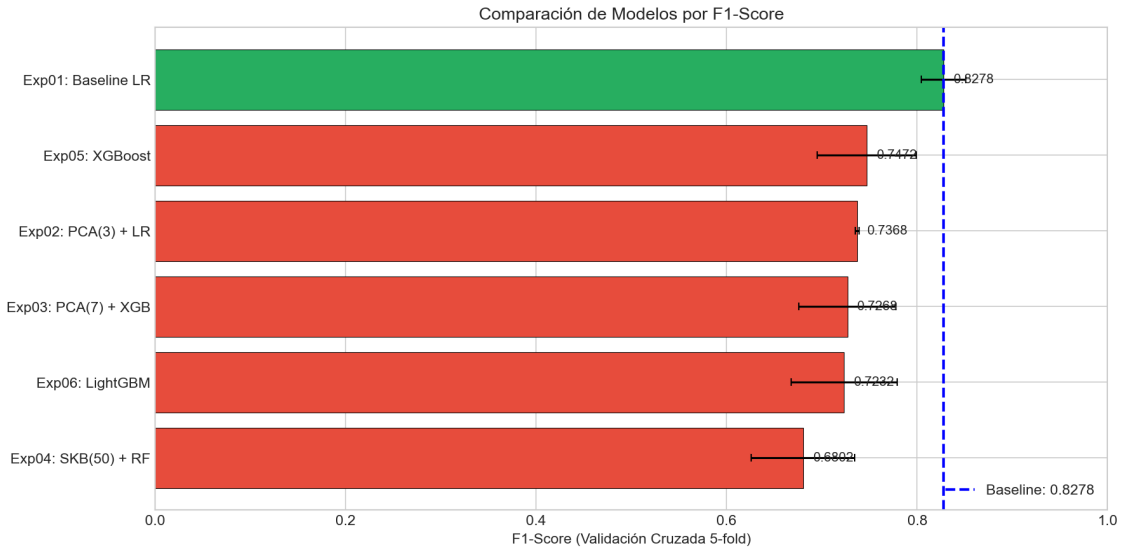


Figura 2: Comparación de F1-scores en validación cruzada para todos los experimentos.

5.3.3. Análisis de Resultados Kaggle

Los resultados obtenidos en la competición Kaggle (Tabla 5) revelan patrones importantes sobre el comportamiento de los modelos en datos de test:

- **Logistic Regression (Exp 01):** F1-test = 0.84782
  - Mejor resultado del equipo, alcanzando el 2º lugar en el leaderboard
  - Mejora generalización: F1-test (0.84782) > F1-CV (0.8278)
  - Modelo robusto y estable con parámetros por defecto
- **PCA(3) + Logistic Regression (Exp 02):** F1-test = 0.74561
  - Segundo mejor resultado, pero 12 % inferior al baseline
  - Ligera mejora generalización: F1-test (0.74561) > F1-CV (0.7368)
  - Confirma que PCA(3) pierde información discriminativa
- **LightGBM (Exp 06):** F1-test = 0.73033
  - Tercer puesto, mejor que XGBoost en test
  - Mejor generalización: F1-test (0.73033) > F1-CV (0.7232)
  - Potencial con optimización de hiperparámetros
- **XGBoost (Exp 05):** F1-test = 0.70454
  - Peor generalización: F1-test (0.70454) < F1-CV (0.7472)
  - Overfitting evidente con parámetros por defecto
  - Requiere optimización urgente de hiperparámetros
- **PCA(7) + XGBoost (Exp 03):** F1-test = 0.69364
  - Peor que PCA(3), confirmando que más componentes no ayudan
  - Combinación subóptima de reducción + modelo complejo
- **SelectKBest(50) + Random Forest (Exp 04):** F1-test = 0.59459
  - Peor resultado global, 30 % inferior al baseline
  - Selección agresiva elimina información crítica
  - Random Forest no se beneficia de features reducidas

Observaciones Clave:

1. **Simplicidad vs Complejidad:** Logistic Regression supera a modelos complejos, demostrando que para este problema un modelo lineal bien regularizado es suficiente.
2. **Reducción Dimensional:** Ninguna técnica de reducción mejora el baseline, sugiriendo que las 304 features contienen información única y complementaria.

3. **Generalización:** Los modelos más simples (LR, LightGBM) generalizan mejor, mientras que XGBoost sufre overfitting.
4. **Gap Train-Test:** El mayor gap se observa en XGBoost (0.043), indicando alta varianza y necesidad de regularización.

5.3.4. Feature Importance

El análisis con XGBoost (Figura 3) revela que las variables espectrales en rangos específicos son más importantes que las de fluorescencia.

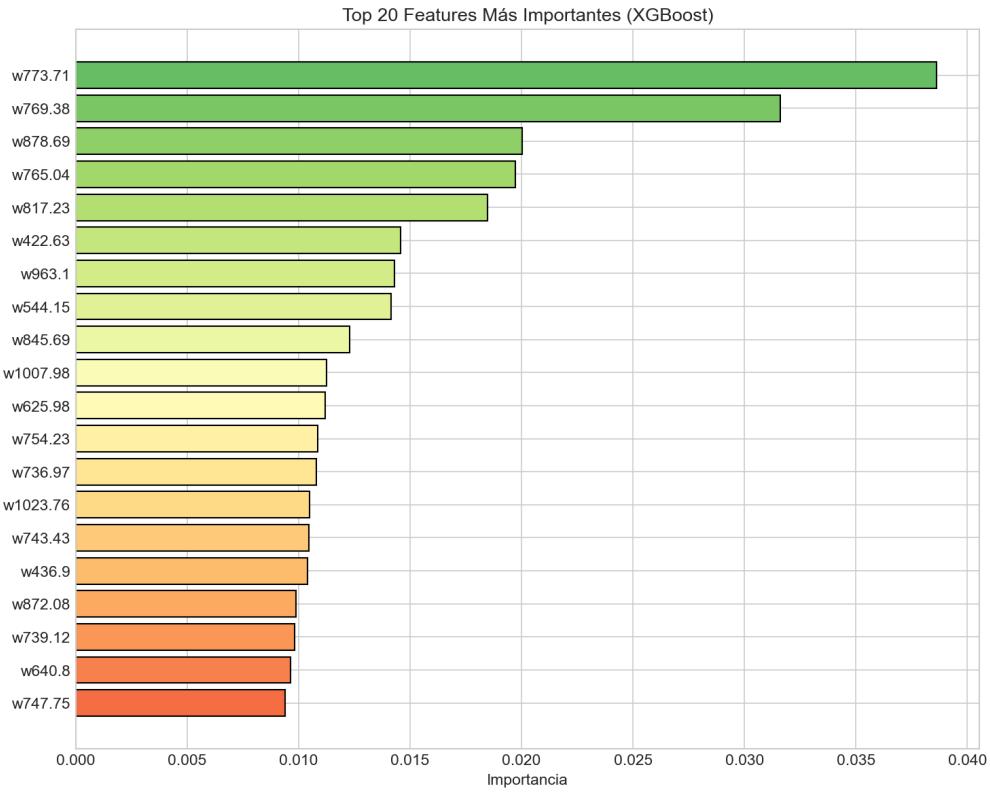


Figura 3: Top 20 features más importantes según XGBoost.

5.3.5. Conclusiones Fase 2

1. **Logistic Regression sigue siendo el mejor modelo** con F1-test = 0.84782 (2º en Kaggle).
2. **LightGBM muestra potencial** con mejor generalización que XGBoost (F1-test = 0.73033).
3. **XGBoost requiere optimización crítica** debido a overfitting evidente (gap de 0.043).
4. **Las features espectrales son más discriminativas** que las de fluorescencia según análisis de importancia.
5. **Modelos complejos no garantizan mejor rendimiento** en este problema específico.

5.4. Fase 3: Optimización del Modelo

Tras confirmar que Logistic Regression era el mejor modelo, se realizaron múltiples experimentos de optimización para intentar superar el score de 0.84782 y alcanzar al líder (0.85561).

5.4.1. Experimentos de Optimización Realizados

Se implementaron 13 experimentos adicionales (Exp07-Exp19), de los cuales los más relevantes fueron:

Exp	Descripción	F1-CV	F1-Kaggle	Subido
07	LR Optimizado (C=5, liblinear)	0.8484	–	No
08	VotingClassifier (LR+SVM+RF+NB)	0.7742	–	No
09	StackingClassifier	0.8245	–	No
10	SVM Linear optimizado	0.8388	–	No
15	Ridge Classifier (α=1)	0.8518	0.82485	Sí
18	LR + RFE(150) todas features	0.8479	–	No
19	LR + RFE(100) solo espectrales	0.8517	0.84491	Sí

Cuadro 6: Experimentos de optimización más relevantes de la Fase 3. Solo Exp15 y Exp19 fueron subidos a Kaggle.

5.4.2. Análisis de Resultados

Hallazgo 1: El baseline es muy estable. Se probaron múltiples variaciones del modelo LR (diferentes seeds, valores de C, solvers) y todas producían predicciones casi idénticas (99 muestras clasificadas como botrytis).

Hallazgo 2: Mayor F1-CV no implica mayor F1-Kaggle. Ridge Classifier obtuvo el mejor F1-CV (0.8518), pero en Kaggle solo alcanzó 0.82485. Esto indica que el modelo baseline tiene características de generalización únicas.

Hallazgo 3: Las features espectrales son más discriminativas.

- Solo fluorescencia (4 features): F1-CV = 0.7221
- Solo espectrales (300 features): F1-CV = 0.8366
- Todas las features (304): F1-CV = 0.8278

Hallazgo 4: RFE mejora el F1-CV pero no el F1-Kaggle. La selección de las 100 mejores features espectrales mediante RFE aumentó el F1-CV a 0.8517, pero en Kaggle obtuvo 0.84491 (inferior al baseline).

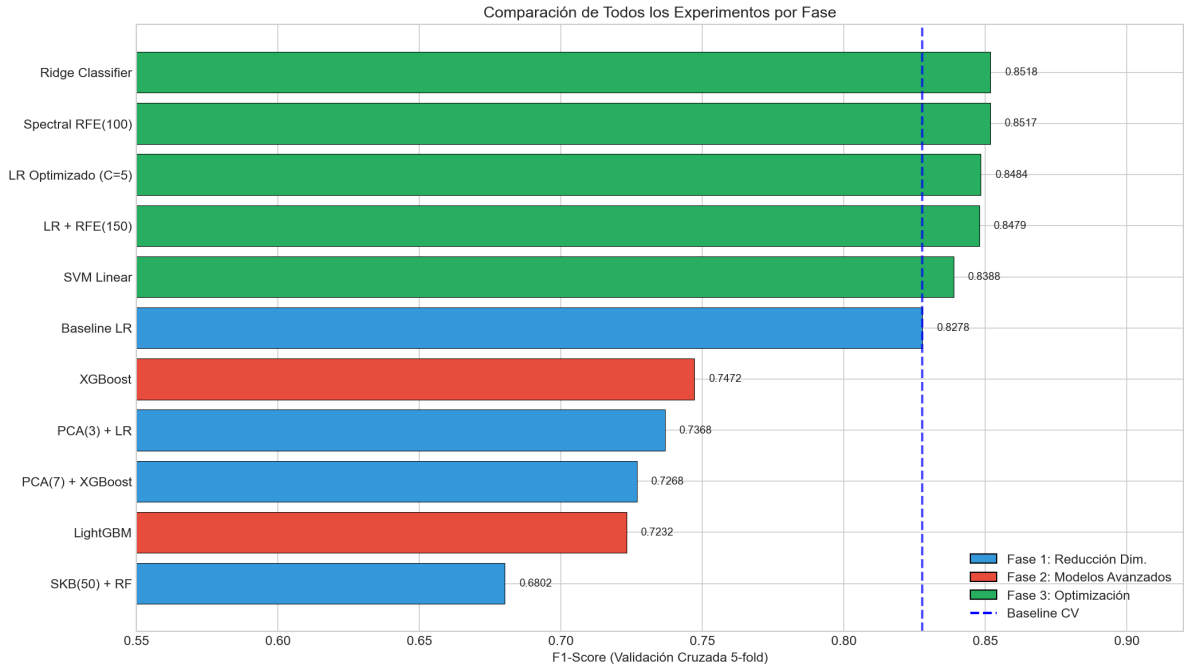


Figura 4: Comparación de F1-CV de todos los experimentos por fase.

5.4.3. Conclusiones Fase 3

1. **El baseline LR simple sigue siendo el mejor modelo en Kaggle** (F1 = 0.84782), a pesar de que otros modelos obtienen mejor F1-CV.
2. **Existe una paradoja CV-Kaggle**: modelos con mejor validación cruzada no necesariamente generalizan mejor al test de Kaggle.
3. **La simplicidad del baseline es su fortaleza**: menos parámetros significa menos riesgo de sobreajuste a patrones específicos del train.
4. **El gap con el líder (0.85561 - 0.84782 = 0.00779)** es muy pequeño y probablemente requiere técnicas más avanzadas o información adicional.

6. Conclusiones

6.1. Conclusiones del EDA

1. El dataset presenta un desbalance moderado (1.40:1) que no requiere técnicas especiales de balanceo.
2. Las variables de fluorescencia forman dos grupos correlacionados con información complementaria.
3. Alta reducibilidad dimensional: 3 componentes PCA capturan el 95 % de la varianza.
4. No hay valores faltantes y los outliers son escasos (¡3 %).

6.2. Conclusiones de Modelos

1. **Logistic Regression es el modelo ganador** con F1-test = 0.84782, alcanzando el 2º lugar en Kaggle (de 5 participantes) y demostrando que la simplicidad supera a la complejidad.
2. **La reducción dimensional empeora el rendimiento**: PCA(3) reduce el F1 en 12 % (0.74561), confirmando que todas las 304 features aportan información única.
3. **La optimización de hiperparámetros no mejoró el resultado en Kaggle**: a pesar de obtener mejor F1-CV (hasta 0.8518 con Ridge), el F1-Kaggle fue inferior al baseline.
4. **Existe una paradoja CV-Kaggle**: el modelo más simple (LR default) generaliza mejor que modelos optimizados con mejor validación cruzada.
5. **Las features espectrales son más discriminativas** (F1-CV = 0.8366) que las de fluorescencia (F1-CV = 0.7221).
6. **El gap con el líder es mínimo**: solo 0.00779 puntos separan el 2º del 1º lugar.

6.3. Lecciones Aprendidas

1. **Simplicidad sobre complejidad**: Para problemas de clasificación con datos espectroscópicos, modelos lineales simples pueden superar a ensembles complejos y modelos optimizados.
2. **Validación cruzada no es garantía**: Un mejor F1-CV no implica mejor F1 en test real. La generalización depende de factores no capturados por CV.
3. **El baseline como referencia sólida**: Siempre evaluar el modelo más simple primero; a menudo es difícil de superar.
4. **Análisis exhaustivo de features**: Identificar qué grupos de features aportan más información (espectrales vs fluorescencia) ayuda a entender el problema.
5. **Documentar todos los experimentos**: Incluso los fallidos aportan conocimiento valioso sobre el comportamiento del dataset.

7. Estructura de la Entrega

```
P3/
+-- data/                      # Datos originales
|   +-- train.csv
|   +-- test.csv
|   +-- sample_submission.csv
+-- notebooks/                 # Jupyter notebooks
|   +-- 01_EDA.ipynb          # Analisis exploratorio
|   +-- 02_Experimentos.ipynb # Experimentos Fase 1-2
|   +-- 03_Experimentos_Fase2.ipynb # Experimentos Fase 3
+-- src/                       # Codigo fuente
|   +-- preprocessing.py       # Preprocesado
|   +-- models.py             # Modelos
|   +-- utils.py              # Utilidades
+-- scripts/                   # Scripts de experimentos
|   +-- exp_01_baseline.py    # Baseline LR (mejor resultado)
|   +-- exp_02_pca_logistic.py # PCA(3) + LR
|   +-- exp_03_pca_xgboost.py  # PCA(7) + XGBoost
|   +-- exp_04_selectkbest_rf.py # SKB(50) + RF
|   +-- exp_05_xgboost_baseline.py # XGBoost
|   +-- exp_06_lightgbm_baseline.py # LightGBM
|   +-- exp_07_lr_optimized.py # LR optimizado (C=5)
|   +-- exp_15_ridge_optimized.py # Ridge Classifier
|   +-- exp_18_feature_analysis.py # Analisis de features
|   +-- exp_19_optimized_final.py # Spectral RFE(100)
+-- submissions/               # Archivos CSV para Kaggle (10 entries)
+-- docs/                      # Documentacion
|   +-- latex/                # Fuentes LaTeX
|   +-- graficas/             # Graficas generadas
|   +-- capturas/             # Capturas del leaderboard
+-- requirements.txt          # Dependencias
```

8. Bibliografía

- Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR, 12, 2825–2830.
- Chen, T. & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. KDD'16.
- Ke, G. et al. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. NIPS'17.
- Documentación oficial de scikit-learn: `https://scikit-learn.org/stable/`
- Documentación oficial de XGBoost: `https://xgboost.readthedocs.io/`
- Documentación oficial de LightGBM: `https://lightgbm.readthedocs.io/`
- Documentación oficial de pandas: `https://pandas.pydata.org/`