APRENDIZAJE AUTOMÁTICO I

Proyecto Práctico II

Boyuan Chen (b.chen@alumnos.upm.es)

Álvaro Huisman (alvaro.huisman@alumnos.upm.es)

INDICE

1. Introduccion:

Se nos ha propuesto realizar una práctica de aprendizaje automático supervisado que debe desarrollarse en Python y usando las librerías de aprendizaje automático vistas en clase. Para ello se nos ha permitido elegir un dataset cual tenga como mínimo 15 variables y 100 observaciones, nosotros hemos elegido un dataset del repositorio de kaggle, y posteriormente a este dataset hemos aplicado las distintas técnicas de aprendizaje automático propuestas que serían SVMs (Máquinas de Soporte Vectorial), K-NN (Vecinos más Cercanos), Árboles de Decisión y Regresión Logística. Se persigue la obtención del modelo óptimo mediante el ajuste de parámetros y la aplicación de diversas estrategias, como la validación cruzada.

1.1 Descripción del problema y objetivo:

Para afrontar esta práctica, hemos escogido un dataset que más se aproxime a nuestros gustos y preferencias, el cual ha sido "Phone Classification Dataset" del repositorio Kaggle, este dataset contiene un total de 21 variables y 2000 filas. Cuyo objetivo es que podamos clasificar móviles en diferentes gamas de precio **(Price range),** siendo 0 la gama más baja y 3 la gama más alta, a partir de las otras variables (variables predictoras).

Variables predictoras:

Nos encontramos con la energía de la batería (**battery_power**), la velocidad del procesador medida en GHz (**clock_speed**), algunos aspectos relacionados con la conectividad, como la presencia de Bluetooth (**blue**), soporte para doble SIM (**dual_sim**), conectividad 4G (**four_g**) y 3G (**three_g**), así como la disponibilidad de WiFi (**wifi**).

En cuanto a la cámara, se incluye la cantidad de megapíxeles tanto de la cámara frontal (**fc**) como de la cámara principal (**pc**). Además, se consideran variables relacionadas con la pantalla, como su resolución en píxeles (**px_height y px_width**), sus dimensiones físicas (**sc_h y sc_w**) y si esta es táctil (**touch_screen**).

El dataset también incluye características sobre la memoria del dispositivo, como la memoria RAM (**ram**) y el almacenamiento interno (**int_memory**), además del número de núcleos del procesador (**n_cores**) y el tiempo de conversación que permite el dispositivo (**talk_time**). Finalmente, se añaden detalles físicos como el peso (**mobile_wt**) y el espesor del dispositivo (**m_dep**).

2. Preprocesamiento:

En el proceso de preprocesamiento, el dataset seleccionado no contenía valores faltantes (**missing values**) y estaba completo a simple vista. Sin embargo, un análisis más profundo reveló que algunas instancias presentaban valores igual a cero en variables como **px_height** y **sc_w**. Esto resulta inconsistente, ya que es improbable que un smartphone no tenga pantalla o que sus dimensiones físicas sean nulas. Por esta razón, decidimos eliminar dichas instancias, reduciendo el dataset a 1819 registros, asegurando así una mayor calidad y veracidad en los datos utilizados.

El **escalado** de los datos fue un paso crucial para garantizar el desempeño de modelos como la **Regresión Logística** y las **Máquinas de Soporte Vectorial (SVM).**

En cuanto a las **SVM**, el objetivo es maximizar el margen entre clases, lo que implica resolver un problema de optimización basado en la distancia de los datos al hiperplano de separación. Por ejemplo, si usamos la distancia Euclidiana:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Si las variables tienen escalas muy diferentes, aquellas con valores más grandes dominarán en los cálculos, sesgando el modelo y afectando el margen óptimo.

De manera similar, en la **Regresión Logística**, la probabilidad de pertenecer a una clase se calcula como:

$$P(y = 1|X) = \frac{1}{1 + e^{-z}}, z = \beta_0 + \sum_{i=1}^{n} \beta_i + x_i$$

Aquí, z es una combinación lineal de las características, y las variables con mayores rangos pueden desbalancear el cálculo de los coeficientes βi, afectando la convergencia del modelo.

Sin embargo, antes de escalar los datos, los dividimos en conjuntos de **train y test**, el escalado debe aplicarse únicamente en los datos de train, manteniendo los datos de test sin alterar para garantizar un análisis realista. Además, algunos modelos, como KNN y árboles de decisión, no son sensibles al escalado o incluso funcionan mejor sin él, por lo que conservamos versiones sin escalar para dichos casos. Para el escalado, utilizamos **MinMaxScaler** (feature_range= (0,1)) en en el conjunto de train, obteniendo valores entre 0 y 1, consistentes con las variables binarias de respuesta. Cabe mencionar que las variables binarias no se escalan para evitar convertir los valores 0 y 1 en floats.

Nota: somos conscientes de que, en un escenario real, el conjunto de datos de prueba (test set) no debería ser accesible durante el desarrollo y entrenamiento del modelo, ya que está reservado para evaluaciones finales del rendimiento en datos completamente desconocidos. Sin embargo, hemos optado por utilizar el conjunto de test en este caso como una herramienta adicional para evaluar el rendimiento de nuestros modelos en nuevas muestras y realizar una comparación más clara entre ellos.

3. Métodos

3.1 Regresión Logística:

En nuestro caso de estudio, no trabajamos con la forma básica de este modelo (regresión logística binaria) ya que tenemos más de dos clases para la variable objetivo, lo que nos lleva a emplear la **regresión logística multinomial**. Este modelo extiende la lógica de la regresión logística binaria a múltiples clases $y \in \{1, 2, \dots, K\}$. La fórmula general es:

$$P(y = k \mid X) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}, donde\ z_k = \beta k0 + \beta k1X1 + \beta k2X2 + \cdots + \beta knXn$$

Vemos que al igual que en su forma binaria es crucial **escalar los datos** debido a la influencia que tienen las diferentes escalas de las variables en los coeficientes estimados.

De hecho, si intentamos entrenar el modelo sin escalarlo nos aparece esta advertencia indicando que el optimizador del modelo no pudo converger al límite de iteraciones establecido.

```
Results without scaling:
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Resultados con escalado:

```
Results with scaling:
C: 100, Mean CV Accuracy: 0.9612
C: 1000, Mean CV Accuracy: 0.9626
C: 10000, Mean CV Accuracy: 0.9648
C: 15000, Mean CV Accuracy: 0.9633
C: 20000, Mean CV Accuracy: 0.9655
C: 30000, Mean CV Accuracy: 0.9663
C: 50000, Mean CV Accuracy: 0.9648
C: 100000, Mean CV Accuracy: 0.9597

Best C with scaling: 30000, Mean CV Accuracy: 0.9663
Metrics on Training Data:
Train Accuracy: 0.9919
Train Precision (Macro): 0.9920
Train Recall (Macro): 0.9919
Train F1 (Macro): 0.9919

Metrics on Test Data:
Test Accuracy: 0.9802
Test Precision (Macro): 0.9804
Test Recall (Macro): 0.9806
Test F1 (Macro): 0.9804
```

```
---------- Test Confusion Matrix
[[106   1   0   0]
 [  1 114   0   0]
 [  0   2 114   3]
 [  0   0   2 112]]

---------- Classification Report ----------
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       107
           1       0.97      0.99      0.98       115
           2       0.98      0.96      0.97       119
           3       0.97      0.98      0.98       114

    accuracy                           0.98       455
   macro avg       0.98      0.98      0.98       455
weighted avg       0.98      0.98      0.98       455

Train Error (directo): 0.0081
Error de prueba Regresión Logística: 0.01978021978021978, con parámetro = 30000
```

Vemos los mejores resultados los obtenemos con C=30000. Este modelo tiene un mean **accuracy** en validación de **0.9663**.

Si analizamos la matriz de confusión de test, observamos que el modelo solo se equivoca en **9 de las 455** instancias. Estos resultados indican que nuestros datos son linealmente separables en gran medida, lo que refuerza la eficacia del modelo aplicado.

3.2 Árboles de decisión

La técnica de **árboles de decisión** funciona dividiendo iterativamente el espacio de datos en subconjuntos más pequeños, utilizando condiciones basadas en las características. Estas divisiones se organizan en una estructura jerárquica similar a un árbol, donde cada nodo representa una pregunta y cada hoja una predicción. El objetivo es maximizar la pureza de las hojas, es decir, que las muestras en cada hoja pertenezcan mayoritariamente a una misma clase.

Como sabemos que no influyen las distancias, creamos el árbol de decisión con los datos sin escalar.

Para obtener el mejor modelo, nos definimos un conjunto de hiperparámetros (`param_grid`) que incluyen aspectos como la profundidad máxima del árbol, el criterio de división y el peso de las clases.

```
param_grid = {
    'min_samples_leaf': [1, 3, 5, 10],
    'min_samples_split': [2, 5, 10, 15],
    'max_depth': [3, 5, 10, 15],
    'criterion': ['entropy', 'gini'],
    'ccp_alpha': [0, 0.001, 0.01],
}
```

Después usamos `GridSearchCV` con validación cruzada (10 particiones) para buscar la combinación de hiperparámetros que maximice la precisión.
Y por último entrenamos el modelo con los datos de entrenamiento y seleccionamos el mejor estimador encontrado, que será utilizado como el modelo final.

```
Best estimator found by grid search:
DecisionTreeClassifier(ccp_alpha=0, criterion='entropy', max_depth=10,
                       min_samples_leaf=5, random_state=42)
CV-Validation Accuracy: 0.8585
```

```
---------------- Classification Report ----------------
              precision    recall  f1-score   support

           0       0.91      0.90      0.90       107
           1       0.82      0.84      0.83       115
           2       0.83      0.75      0.79       119
           3       0.85      0.91      0.88       114

    accuracy                           0.85       455
   macro avg       0.85      0.85      0.85       455
weighted avg       0.85      0.85      0.85       455
```

```
---------------- Test Confusion Matrix
[[ 96  11   0   0]
 [ 10  97   8   0]
 [  0  11  89  19]
 [  0   0  10 104]]

  Metrics on Training Data:
  Train Accuracy: 0.9597
  Train Precision (Macro): 0.9596
  Train Recall (Macro): 0.9596
  Train F1 (Macro): 0.9596

  Metrics on Test Data:
  Test Accuracy: 0.8484
  Test Precision (Macro): 0.8495
  Test Recall (Macro): 0.8502
  Test F1 (Macro): 0.8489
```

```
Train Error (directo): 0.0403
Error de prueba Árbol de decisión: 0.1516, con parámetros: DecisionTreeClassifier(ccp_alpha=0, criterion='entropy', max_depth=10,
                       min_samples_leaf=5, random_state=42)
```

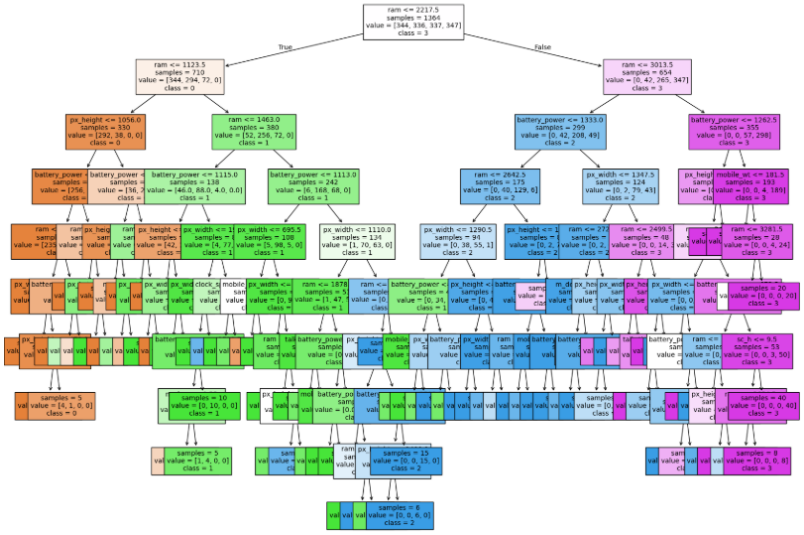
Vemos que el mejor modelo de árbol de decisión encontrado es el modelo con ccp_alpha = 0, criterio = ‘entropy’, max_depth = 10 y min_samples_leaf = 5.

Observamos que el **Mean CV Accuracy** obtenido durante la validación cruzada es considerablemente inferior al logrado por el mejor modelo de la técnica anterior. Además, esto podría deberse a un posible **sobreentrenamiento**, ya que el accuracy en el conjunto

de entrenamiento es notablemente más alto. Este comportamiento podría ser causado por un modelo que se adapta excesivamente a los datos de entrenamiento, capturando patrones específicos o ruido que no generalizan bien a datos nuevos. Por tanto, probamos ajustar hiperparámetros como limitar el max_depth o incrementar min_samples_leaf y ccp_alpha para reducir el sobreentrenamiento. Sin embargo, el modelo de árbol de decisión sigue mostrando un rendimiento inferior en el conjunto de prueba, indicando problemas de generalización. Por ello, concluimos que este modelo no es el más adecuado para nuestros datos.

Para entender mejor esto observamos las reglas generadas por nuestro Árbol de decisión:

```
Profundidad del arbol: 9
Número de nodos terminales: 79
```



Podemos ver que el árbol comienza con `ram` como la variable más importante en la raíz (`ram <= 2117.5`). Cada nodo representa una regla de decisión que separa los datos en subconjuntos más homogéneos en términos de clase, evaluando variables como `battery_power`, `px_height` y `mobile_wt`. Las hojas terminales contienen la predicción final basada en la clase mayoritaria de las muestras que llegan allí. Con 9 niveles de profundidad y 79 hojas terminales, el árbol tiene reglas detalladas, pero es bastante complejo.

3.3 K-NN

KNN funciona identificando los **k vecinos más cercanos** a un punto de datos basándose en una métrica de distancia (como la euclidiana, manhattan...) en el espacio de características. Para clasificación, asigna la clase mayoritaria entre los vecinos su desempeño depende de elegir un valor óptimo de k.

En nuestro caso hemos usado los datos sin escalar ya que al escalar los datos todas las características contribuyen de manera equitativa al cálculo de las distancias y quitarían importancia a las variables con más peso como por ejemplo ram o battery_power.

Para obtener el mejor modelo nos creamos un conjunto de hiperparámetros (`param_grid`) que incluyen aspectos como el rango de Ks y las distintas métricas de distancias.

```
# Rango de valores para k y métricas de distancia
k_range = list(range(1, 21))
metrics = ['euclidean', 'manhattan', 'chebyshev', 'minkowski']
param_grid = {'n_neighbors': k_range, 'metric': metrics}
```

```
---------- Test Confusion Matrix
[[104   3   0   0]
 [  3 103   9   0]
 [  0   5 105   9]
 [  0   0   7 107]]
```

```
CV-Validation Accuracy: 0.9361957921854872
Mejor K: 9, Mejor Métrica: euclidean
```

```
Metrics on Training Data:
Train Accuracy: 0.9494
Train Precision (Macro): 0.9492
Train Recall (Macro): 0.9492
Train F1 (Macro): 0.9492

Metrics on Test Data:
Test Accuracy: 0.9209
Test Precision (Macro): 0.9225
Test Recall (Macro): 0.9221
Test F1 (Macro): 0.9222
```

```
---------- Classification Report ----------
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       107
           1       0.93      0.90      0.91       115
           2       0.87      0.88      0.88       119
           3       0.92      0.94      0.93       114

    accuracy                           0.92       455
   macro avg       0.92      0.92      0.92       455
weighted avg       0.92      0.92      0.92       455
```

```
Train Error (directo): 0.0506

Error de prueba KNN: 0.0791, con K = 9 y Métrica = euclidean
```

Vemos que el cross validation nos muestra como mejor modelo un KNN con K =9 usando la distancia euclidea, alcanzando una precisión media en validación cruzada (CV) de **0.93165.** El mejor modelo, usando los parámetros obtenidos anteriormente da ante los datos de test solamente **36 resultados erróneos de un total de 455 datos** lo que es relativamente bajo. El error de prueba es bajo (**0.0632**) y comparable al error en entrenamiento (**0.0685**), lo que sugiere que el modelo generaliza bien y no está sobre ajustado.

3.4 SVM

Como mencionamos en el preprocesamiento, las Máquinas de Soporte Vectorial (SVM) son modelos supervisados que buscan maximizar el margen entre clases en los datos, encontrando un hiperplano óptimo que separe las categorías. Este modelo es particularmente útil en problemas linealmente separables, aunque también puede utilizar kernels no lineales para trabajar con datos más complejos. Además, como dijimos antes, es fundamental escalar los datos, ya que las SVM son sensibles a la magnitud de las variables.

En la regresión logística observamos que los datos eran linealmente separables, lo que sugiere que el kernel **lineal** es el más adecuado para este problema. Sin embargo, para confirmar esta elección, realizaremos un pequeño grid search que incluirá otros kernels, como **polinomial** y **RBF**, y probaremos varios valores del hiperparámetro C. Esto nos permitirá validar que el kernel lineal proporciona el mejor rendimiento en este caso.

```
# Definir el rango ampliado de hiperparámetros a probar
param_grid = {
    'C': [100, 1000, 10000,50000,75000, 100000],
    'kernel': ['linear', 'poly', 'rbf'],
}
```

```
---------------- Test Confusion Matrix -
[[105   2   0   0]
 [  1 114   0   0]
 [  0   2 111   6]
 [  0   0   1 113]]
```

```
Best parameters found by grid search: {'C': 50000, 'kernel': 'linear'}
CV-Validation Accuracy: 0.9655324173465006
```

```
Metrics on Training Data:
Train Accuracy: 0.9978
Train Precision (Macro): 0.9978
Train Recall (Macro): 0.9978
Train F1 (Macro): 0.9978

Metrics on Test Data:
Test Accuracy: 0.9736
Test Precision (Macro): 0.9743
Test Recall (Macro): 0.9742
Test F1 (Macro): 0.9739
```

```
---------------- Classification Report ----------------
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       107
           1       0.97      0.99      0.98       115
           2       0.99      0.93      0.96       119
           3       0.95      0.99      0.97       114

    accuracy                           0.97       455
   macro avg       0.97      0.97      0.97       455
weighted avg       0.97      0.97      0.97       455
```

```
Train Error (directo): 0.0022

Error de prueba SVM: 0.0264, con parámetros: SVC(C=50000, kernel='linear')
```

El resultado del grid search confirma que el **kernel lineal** es el más adecuado para este problema, obteniendo los mejores resultados con C=50000. Esto es consistente con lo observado previamente en la regresión logística, donde determinamos que los datos eran linealmente separables. Este modelo además alcanzó un **mean accuracy de validación** de **0.9655**.

En cuanto a su funcionamiento antes los datos de test, vemos, al analizar la matriz de confusión, que el modelo solo comete **12 errores en las 455 instancias**, evidenciando su alta precisión y robustez. Estos resultados refuerzan que el kernel lineal no solo es el más simple, sino también el más efectivo para este conjunto de datos, evitando la complejidad innecesaria de otros kernels, como el polinomial o el RBF, que no aportan mejoras significativas en este caso.

4. Comparativa de resultados y Conclusiones

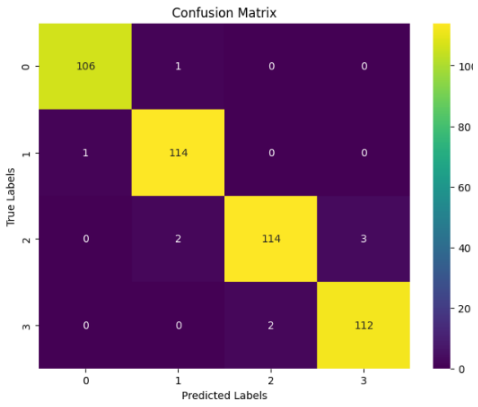
Aunque hemos usado el test para comparar el rendimiento de diferentes modelos ante datos nuevos, cabe recalcar que la selección del modelo final no se basa en este rendimiento, sino en los valores de **Mean CV Accuracy** obtenidos durante la validación cruzada. Este enfoque es más representativo de un entorno real, ya que garantiza que la evaluación y selección de los modelos se realicen exclusivamente sobre los datos de entrenamiento y validación, sin comprometer la independencia del conjunto de prueba.

CROSS VALIDATION				
	Regresión Logística	**Árbol de Decisión**	**KNN**	**SVM**
Mean CV Accuracy	0.9663	0.8585	0.9361	0.9655

TABLA DE COMPARACIÓN (resultados test)				
	Regresión Logística	Árbol de Decisión	KNN	SVM
Accuracy	0.9802	0.8484	0.9209	0.9736
Precision (Macro)	0.9804	0.8495	0.9225	0.9743
Recall (Macro)	0.99806	0.8502	0.9221	0.9742
F1 (Macro)	0.9804	0.8489	0.9222	0.9739
Error validación	0.0198	0.1516	0.0506	0.0264

De acuerdo con los resultados de **Mean Accuracy** en validación, observamos que hay dos modelos que destacan sobre el resto: la Regresión Logística y SVM. Aunque podríamos optar por cualquiera de los dos, ya que muestran métricas muy similares, hemos elegido el modelo de **Regresión Logística** con el parámetro **C=30000** debido a su ligera superioridad en las métricas, su simplicidad y la rapidez de su entrenamiento.

Y obtenemos como **resultados finales**:




```
---------------- Classification Report ----------------
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       107
           1       0.97      0.99      0.98       115
           2       0.98      0.96      0.97       119
           3       0.97      0.98      0.98       114

    accuracy                           0.98       455
   macro avg       0.98      0.98      0.98       455
weighted avg       0.98      0.98      0.98       455

Metrics on Test Data:
Test Accuracy: 0.9802
Test Precision (Macro): 0.9804
Test Recall (Macro): 0.9806
Test F1 (Macro): 0.9804
```

```
Error de prueba Regresión Logística: 0.019780219780219778, con parámetro = 30000
```

El modelo final (regresión logística) demuestra un rendimiento sobresaliente, logrando una precisión del 98.02% en las predicciones. Las métricas de precisión, recall y F1-score son consistentemente altas para todas las clases, con valores cercanos a 0.98. Esto indica un buen equilibrio en la identificación correcta de las clases y la minimización de errores. Además, el error de prueba es extremadamente bajo (0.01978), lo que sugiere que el modelo está bien ajustado y no presenta sobreajuste, mostrando un excelente desempeño en la clasificación de las distintas gamas de precio.

Es importante señalar que los errores de clasificación se limitan principalmente a una gama de precio más alta o más baja, lo que no es particularmente problemático dado el contexto. Esto confirma que el modelo realiza una clasificación precisa y confiable, con errores mínimos que no afectan significativamente su utilidad.