Cierre

A modo de *photo finish* hago aquí un compendio de los puntos claves que habría que tener en cuenta en cada uno de los puntos que hemos estado viendo

---

🧠 1. Fundamentos Clave de Machine Learning

🔷 Flujo de trabajo de un proyecto de ML

Este es el esquema básico que se debe aplicar en cualquier problema de IA:

1. **Análisis Exploratorio de Datos (EDA)**
   - Inspección de datos: `df.info()`, `df.describe()`, valores nulos, *outliers*.
   - Visualizaciones básicas: histogramas, boxplots, correlaciones.
2. **Preprocesamiento**
   - **Limpieza**: eliminar o imputar valores nulos.
   - **Normalización o estandarización:** especialmente para modelos sensibles a escala (SVM, KNN, redes).
   - **Codificación:** `LabelEncoder`, `OneHotEncoder`, etc.
3. **División del dataset**
   - `train_test_split()` → evitar sesgos.
   - Ideal: usar también validación cruzada (`KFold`, `StratifiedKFold`).
4. **Entrenamiento y evaluación**
   - Entrenar múltiples modelos, comparar con métricas.
   - Evitar *overfitting* con regularización y validación cruzada.
5. **Persistencia**
   - Guardar modelos con `joblib`, `pickle` o `model.save()` (Keras).

📊 2. Métricas de Evaluación por tipo de problema

🔶 Clasificación

- **Accuracy**: proporción de aciertos. Cuidado con clases desbalanceadas.
- **Precision**: cuántos de los positivos predichos son correctos.
- **Recall**: cuántos de los positivos reales fueron encontrados.
- **F1-score**: equilibrio entre precision y recall.
- **ROC-AUC**: ideal para modelos binarios.

✅ Saber cuándo priorizar recall (detección de enfermedades) vs precision (filtros de spam).

🔶 Regresión

- **MSE (Error cuadrático medio)**: penaliza más los errores grandes.
- **MAE (Error absoluto medio)**: más robusto a outliers.
- **R^2 (Coeficiente de determinación)**: proporción de varianza explicada por el modelo.

⚙️ 3. Modelado Supervisado

🔶 Modelos clásicos

Cuándo usar cada uno:

Modelo	Ideal para...
Regresión Lineal	Relación lineal entre variables.
Regresión Logística	Clasificación binaria.
KNN	Datos sin muchas dimensiones, no lineales.
Árboles de decisión	Interpretabilidad, rapidez.
Random Forest	Precisión alta, manejo de *overfitting*.
SVM (Máquina de vectores de soporte)	Datos linealmente separables, márgenes claros.

👉 Saber cuáles son sensibles a escala (SVM, KNN) y cuáles no (árboles).

🔶 Pipelines

- Automatiza el preprocesamiento + entrenamiento.
- Ejemplo con `Pipeline` y `ColumnTransformer` de `sklearn`.

```
pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("model", RandomForestClassifier())
])
```

🔶 Hiperparámetros

- Usar `GridSearchCV` o `RandomizedSearchCV` para buscar valores óptimos.
- Enseñar el concepto de *cross-validation* para estimación robusta.

🔍 4. Modelado No Supervisado

🔶 Clustering

- **K-means**: rápido, requiere elegir `k`. Usar *elbow method*, *Silhouette score*.
- **DBSCAN**: útil para detectar ruido y clusters de forma irregular.
- **Clustering jerárquico**: bueno para entender estructura de grupos.

🔶 Reducción de dimensionalidad

- **PCA**: reducir sin perder mucha varianza. Útil antes de clustering.
- **t-SNE / UMAP**: solo para visualización de alta dimensión.

✅ Entender la diferencia entre *técnicas de reducción* y *clustering*.

🤖 5. Redes Neuronales

🔶 Arquitectura densa (MLP)

- Capas densas (`Dense`), activaciones (`ReLU`, `sigmoid`, `softmax`).
- *Overfitting* frecuente: usar `Dropout`, `L2 regularization`.

```
model = Sequential([
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(10, activation='softmax')
])
```

🔶 Entrenamiento

- Funciones de pérdida: `categorical_crossentropy`, `binary_crossentropy`, `mse`.
- Optimizadores: `SGD`, `Adam` (el más usado).
- Callbacks: `EarlyStopping`, `ModelCheckpoint`.

🔶 Evaluación

- Asegurarse de evaluar en datos no vistos.
- Visualizar curvas de pérdida/accuracy.

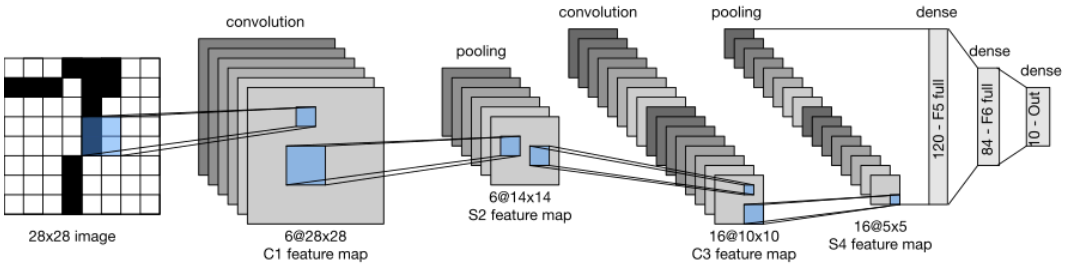
🧱 6. Redes Convolucionales (CNN)

🔶 Componentes clave

- **Convoluciones**: extracción de patrones espaciales.
- **Pooling**: reducción de dimensionalidad.
- **Flatten + Dense**: capa final para clasificación.

🔶 Arquitecturas simples

- LeNet, VGG16 (capas secuenciales).



- Saber qué hacen los filtros, cómo afectan al tamaño del output.

🔶 Dataset y preprocesado

- Usar `ImageDataGenerator` para aumento de datos (data augmentation).
- Imágenes deben estar escaladas ([0, 1] o [-1, 1]).

🔁 7. Transfer Learning

🔶 Cuándo usarlo

- Cuando no se dispone de un dataset grande.
- Para tareas similares a las del modelo preentrenado.

🔶 Técnicas

- **Feature Extraction**: congelar capas convolucionales, solo cambiar el "cabezal".
- **Fine Tuning**: descongelar algunas capas y reentrenar con tasa de aprendizaje baja.

```
base_model.trainable = False  # feature extraction
```

🔶 Modelos típicos

- `VGG16`, `ResNet50`, `EfficientNet`, `MobileNet`.

🛠️ 8. Habilidades prácticas que deberían adquirir

- Uso fluido de Scikit-learn, Pandas, Matplotlib/Seaborn.
- Saber cómo estructurar un cuaderno de Jupyter de principio a fin.
- Documentar decisiones y resultados con claridad.
- Buenas prácticas: modularidad, reproducibilidad, evitar hardcoding.

🚀 9. Mentalidad final

- No se trata de saberse todos los modelos, sino de:
  - Elegir la herramienta adecuada.
  - Justificar cada decisión.
  - Aprender a depurar errores y mejorar modelos iterativamente.
- La intuición y el sentido común son tan importantes como la técnica.