# PROYECTO DE MACHINE LEARNING

## PARTE 1: APRENDIZAJE SUPERVISADO (PYTHON) - Análisis de Fraude en Transacciones con Tarjetas de Crédito

#### DESCRIPCIÓN GENERAL

En este proyecto, aplicarás técnicas de Machine Learning para detectar si una transacción de tarjeta de crédito es fraudulenta o no. Trabajarás con un conjunto de datos real, transformado para proteger la confidencialidad de la información. Este proyecto pondrá a prueba tus habilidades en exploración de datos, preprocesamiento, selección de modelos, evaluación y presentación de resultados.

El objetivo es construir un modelo que pueda clasificar las transacciones en fraudulentas o no fraudulentas, enfrentando el desafío de un conjunto de datos altamente desbalanceado.

#### Contexto de los Datos

El conjunto de datos, llamado **creditcard.csv**, incluye información de transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares europeos. Contiene:

* Presenta transacciones que ocurrieron en dos días**, 492 transacciones fraudulentas** de un total de **284,807 transacciones**.
* El conjunto de datos está **altamente desbalanceado**, ya que la clase positiva (fraudes) **representa solo el 0.172% de todas las transacciones**.
* Contiene **únicamente variables numéricas**, **transformadas mediante PCA:**
  + Por cuestiones de confidencialidad, no se pueden proporcionar las características originales ni más información de contexto sobre los datos.
  + Las características **V1, V2, … V28** son los componentes principales obtenidos con PCA
* **Dos variables adicionales no transformadas** con PCA:
  + *Time*: Segundos transcurridos entre cada transacción y la primera transacción en el conjunto de datos
  + *Amount*: Monto de la transacción.
* **La variable objetivo** es*Class*:
  + 1 = Fraude.
  + 0 = No fraude.

#### Objetivos del Proyecto

1. Construir un pipeline completo de Machine Learning para clasificar transacciones fraudulentas.
2. Justificar cada decisión tomada en el preprocesamiento, selección de modelos y evaluación.
3. Manejar el desbalanceo de clases de manera efectiva.
4. Comparar y optimizar diferentes modelos.
5. Presentar un script final que resuma la solución implementada.

#### Pasos a seguir:

* Obtener los datos
* Explorar los datos
* Preparar los datos
* Entrenar y evaluar en el conjunto de entrenamiento con distintos modelos (prueba 4 modelos y luego un ensamblaje de los cuatro). ¿Qué métrica vas a usar? ¿Por qué?
* Optimizar el mejor modelo del paso anterior
* Evaluar en el conjunto de pruebas
* Guardar el modelo
* Presentar la solución (script aparte).
* Buenas practicas:
  + **Comentarios en el código:** Comentar las decisiones o pasos relevantes en el código
  + **Incluir docstrings:** Documentar las funciones dentro del script para facilitar el entendimiento con la librería docstrings.
  + **Añadir logs de control:** Usar el módulo logging para registrar pasos importantes durante la ejecución y los errores que puedan darse.

#### **Formato de Entrega**

1. **Estructura de Carpetas** El proyecto debe seguir esta estructura:

**3.3\_Proyecto\_ML\_GX**/

│

├── datos/ # Carpeta con los datos

│ └── creditcard.csv # Datos con los que trabajar

├── recursos/ # Recursos adicionales, imágenes, modelos

│ ├── imagenes/ # Carpeta de imagenes

│ │ ├── imagen1.png

│ │ └── imagen2.png

│ └── modelos/ # Modelos creados

│ └── trans\_fraudulenta.joblib # Modelo obtenido de la parte 1

├── src/ # Carpeta con codigo fuente del proyecto

│ └── parte1\_AS/ # Carpeta de código de Parte 1

│ ├── notebooks/ # Cuadernos de Jupyter

│ └── resultados.py # Script para la presentación de resultado

├── logs/ # Registros de ejecución y errores

│ ├── proc\_ejecucion.log # log de proceso de ejecución

│ └── errores.log # log de errores

└── documentacion/ # Explicación detallada del proyecto

│ ├── README.md # Cuadernos de Jupyter

│ └── ayuda.html # HTML generado de docstring

└── .gitignore # Archivo de git para evitar ficheros

1. **Documentación** Incluye un archivo README.md en la carpeta documentacion/ con:
   * Descripción general del proyecto.
   * Breve explicación del enfoque utilizado.
   * Instrucciones para ejecutar el proyecto.
2. **Justificación** En cada script, incluye comentarios y documentación (docstrings) explicando cada paso.

**NOTA: ES MUY IMPORTANTE JUSTIFICAR Y EXPLICAR TODO LO QUE HACES**

## PARTE 2: APRENDIZAJE NO SUPERVISADO (PYTHON) - Clustering de Caras con K-Means y PCA

#### DESCRIPCIÓN GENERAL

En este proyecto, se aplicarán técnicas de reducción de dimensionalidad y clustering para agrupar imágenes de rostros. El objetivo principal es identificar similitudes entre caras y agruparlas, de modo que al recibir una nueva imagen, podamos predecir a qué grupo pertenece.

#### Contexto de los Datos

El conjunto de datos olivetti\_faces.csv contiene:

* **400 imágenes de 64x64 píxeles en escala de grises**, de 40 personas sin etiquetar
* Cada imagen se aplana a un vector 1D de tamaño 4.096
* Las imágenes no están etiquetadas (es un problema de aprendizaje no supervisado).
* La tarea principal es agrupar las imágenes en grupos similares.

**Nota:** Una vez elegido el mejor modelo, se utilizará para predecir el grupo al que pertenece una nueva imagen (olivetti\_face\_sample.png).

#### Pasos a seguir:

* Cargar los datos de olivetti\_faces.csv
* Aplica PCA a X para reducir la dimensionalidad, pero conservando el 99% de la varianza. ¿Cuántas componentes quedan?
* Agrupa las imágenes utilizando k-meas ¿piensa que valor de vas a dar a k y por qué?
* Visualiza los grupos: ¿ves caras similares en cada grupo?

Saca conclusiones

* Agrupa de nuevo las imágenes utilizando k-means, pero esta vez utiliza las puntuaciones de la silueta para elegir cual es el número adecuado de centros (quiero ver la gráfica).
* Visualiza los grupos: ¿ves caras similares en cada grupo?

Saca conclusiones

* ¿Qué pasa si repetimos lo mismo (k-means eligiendo la mejor k con las puntuaciones de la silueta), pero sin aplicar PCA?:
* ¿Tarda más o menos? ¿Los resultados son mejores, peores o parecidos?
* Saca conclusiones
* Decídete por un modelo (obviamente el mejor de los que has probado), en caso de aplicar PCA, crea un modelo que aplique PCA y K-Means, entrena el modelo y guarda el modelo en un fichero.
* En un script (resultado\_ANS.py) importa el modelo importa la imagen olivetti\_face\_sample.png y predice a que clúster pertenece:
* El programa debe comprobar si existe el archivo del modelo .joblib. Si no se encuentra, debe notificar al usuario y salir.
* Solicita el nombre del archivo de la imagen que quieres predecir. Si el archivo no existe, el programa debe seguir pidiendo el nombre hasta que el archivo proporcionado sea válido.
* Maneja cualquier error que pueda ocurrir, como problemas de lectura del modelo o de la imagen.
* Buenas prácticas:
  + **Comentarios en el código:** Comentar las decisiones o pasos relevantes en el código
  + **Incluir docstrings:** Documentar las funciones dentro del script para facilitar el entendimiento con la librería docstrings.
  + **Añadir logs de control:** Usar el módulo logging para registrar pasos importantes durante la ejecución y los errores que puedan darse.

Cuando trabajemos con redes neuronales los resultados serán mucho mejores.

#### **Formato de Entrega**

1. **Estructura de Carpetas** El proyecto debe seguir esta estructura:

**3.3\_Proyecto\_ML\_GX**/

│

├── datos/ # Carpeta con los datos

│ ├── creditcard.csv # Datos con los que trabajar parte 1

│ └── olivetti\_faces.csv # Datos con los que trabajar parte 2

├── recursos/ # Recursos adicionales, imágenes, modelos

│ ├── imagenes/ # Carpeta de imagenes

│ │ ├── olivetti\_face\_sample.png # Imagen de parte 2

│ │ └── imagen2.png

│ └── modelos/ # Modelos creados

│ ├── trans\_fraudulenta.joblib # Modelo obtenido de la parte 1

│ └── modelo\_kmeans125.joblib # Modelo obtenido de la parte 2

├── src/ # Carpeta con codigo fuente del proyecto

│ └── parte1\_AS/ # Carpeta de código de parte 1(AS)

│ ├── notebooks/ # Cuadernos de Jupyter

│ └── resultado\_AS.py # Script para la presentación de resultado

│ └── parte2\_ANS/ # Carpeta de código de parte 2(ANS)

│ ├── notebooks/ # Cuadernos de Jupyter

│ └── resultado\_ANS.py # Script para la presentación de resultado

├── logs/ # Registros de ejecución y errores

│ ├── proc\_ejecucion.log # log de proceso de ejecución

│ └── errores.log # log de errores

└── documentacion/ # Explicación detallada del proyecto

│ ├── README.md # Cuadernos de Jupyter

│ └── ayuda.html # HTML generado de docstring

└── .gitignore # Archivo de git para evitar ficheros

1. **Documentación** Incluye un archivo README.md en la carpeta documentacion/ con:
   * Descripción general del proyecto.
   * Breve explicación del enfoque utilizado.
   * Instrucciones para ejecutar el proyecto.
2. **Justificación** En cada script, incluye comentarios y documentación (docstrings) explicando cada paso.

**NOTA: ES MUY IMPORTANTE JUSTIFICAR Y EXPLICAR TODO LO QUE HACES**

## PARTE 3: USO DE CONTROL DE VERSIONES Y BUENAS PRÁCTICAS

#### **control de versiones**

La implementación del **control de versiones** en el proyecto de Machine Learning es obligatorio para asegurar que el equipo colaborare de manera efectiva y mantener un historial claro de todos los cambios realizados.

A cada equipo de trabajo se le asignara

* Un repositorio privado creado con el nombre **3.3\_Proyecto\_ML\_GX** donde X sea el numero de grupo asignado.
* El repositorio contiene un proyecto asociado con un tablero Kanban en el que introduciremos las tareas a realizar.

Todos los integrantes del grupo deberán de utilizar git de forma activa durante las semanas que dure el proyecto, de forma, que si se revisa el histórico se aprecie el trabajo de ambos integrantes en ambas partes del proyecto.

##### **contenido de github**

1. **Repositorio** 3.3\_Proyecto\_ML\_GX: Estructura del proyecto
2. **Proyecto** 3.3\_Proyecto\_ML\_GX: Tablero Kanban
3. **Issues** del proyecto 3.3\_Proyecto\_ML\_GX: Gestión de tareas
4. **Pull Request en el repositorio** 3.3\_Proyecto\_ML\_GX: Los merges a la rama main serán mediante pull request que tendrá que ser aprobada por el compañero. Revisando que el contenido a subir el correcto.

##### **Estructura de ramas**

1. **Rama principal (main)**: Contendrá la versión estable del proyecto.
2. **Ramas individuales**: Cada estudiante crea una rama para trabajar. Cada vez que termine una de las fases, deberá de integrar el código con la rama principal.
   * **Nombre de las ramas**: feature/alumnos1, feature/parte1, feature/creación\_py
   * **Nombre de ramas si se detectan errores en la rama** main: bugfix/error\_analisis…

##### **Flujo de trabajo recomendado:**

1. **Clona el repositorio creado en GitHub 3.3\_Proyecto\_ML\_GX** en tu local para trabajar.
2. **La rama principal de trabajo será** main
3. **Cada integrante del grupo deberá de crear y trabajar en una rama. Ya sea la misma rama para cada parte de trabajo realizado o una rama con la funcionalidad concreta que se va a hacer. Decisión por parte del equipo.**
4. **Cuando se termine una funcionalidad (análisis de datos, división, entrenamiento…), se deberá fusionar con la rama principal.**
5. **Crear un Pull Request para la integración de las ramas de los alumnos con** main**:**
   * Sube los cambios y envía un **Pull Request** para fusionar la rama en main.
   * Asegúrate de que otro estudiante revise el PR antes de aceptarlo.
6. Integrar los cambios en la rama main(Es lo que se va a corregir)

##### **BUENAS PRÁCTICAS EN GIT (OBLIGAtorio)**

1. **Mensajes de commits claros:** Ejemplo:
   * ✅ Añadido análisis de PCA con reducción a 2 dimensiones.
   * ✅ Corrección en los parámetros de KMeans para silueta óptima.
   * ❌ Cambios realizados. (Evitar mensajes vagos).
2. **Revisión por pares:**  
   Cada PR debe ser revisado por otro estudiante para verificar:
   * La claridad del código.
   * Evitar errores no detectados previamente
3. **Validación Frecuente:**
   * Son necesarios los commits frecuentes para poder tener un histórico de los cambios realizados en el proyecto.
   * El avance del proyecto se tiene que ver en el histórico de git.
4. **Evitar commits grandes y generales:**
   * El trabajo no será evaluado si solo se hace un commit con todos los cambios o no se ve una continuidad.
   * Todos los integrantes del grupo de trabajo deben participar activamente en todas las partes del trabajo.

#### **buenas prácticas – tareas a realizar**

##### **Documentación en funciones Y COMENTARIOS EN CODIGO**

La documentación en las funciones de Python es fundamental para mantener un código limpio, entendible y fácil de mantener. En Python, se utiliza un formato especial llamado **docstring** para documentar funciones, clases y módulos.

Es obligatorio incluir en el código:

* **Comentarios de código** - **es obligatorio incluir comentarios** en las siguientes partes del proyecto:
  + **Descripción general del código**: Al inicio de cada archivo Python, debe haber un comentario de cabecera que explique de manera breve el propósito general del script o módulo. Esto debe incluir:
    - Qué hace el script en términos generales.
    - Qué tipo de datos se procesan.
    - El objetivo del código (por ejemplo, entrenar un modelo de Machine Learning, realizar análisis exploratorio, etc.).
  + **Comentarios en funciones**: Cada función debe estar documentada utilizando un docstring (comentarios entre triples comillas """).
    - **Descripción breve** (línea 1): Es una descripción concisa de lo que hace la función.
    - **Parámetros**: Una lista de los parámetros de entrada, con una breve descripción de cada uno. El tipo de dato también se debe indicar.
    - **Valor de retorno**: Describe lo que la función devuelve (si devuelve algo), junto con el tipo de retorno.
    - **Excepciones (opcional):** Si la función puede lanzar excepciones, es una buena práctica documentarlas.
  + **Comentarios en bloques de código importantes**: Es importante comentar las secciones del código que involucran decisiones clave, como la selección de hiperparámetros en un modelo de Machine Learning o transformaciones de datos.
* **Generar documentación en base a docstrings (Herramienta):**
  + **pdoc**: Herramienta ligera para generar documentación en HTML desde docstrings.

##### **Logging para seguimiento**

Implementar **logging** en proyectos de **Machine Learning** es esencial para hacer un seguimiento detallado de las operaciones que se realizan durante el ciclo de vida del modelo, desde la carga de datos hasta la evaluación del modelo.

Se van a crear dos ficheros de logging:

* proc\_ejecucion.log: Log que irá añadiendo los procesos realizados durante la ejecución
* errores.log: Lo que mostrará la información de errores sufridos durante la ejecución.

En el caso del log del proceso de ejecución, agrega registros para documentar el progreso y facilitar la depuración:

1. Instalación y Configuración Básica
2. Carga y Preprocesamiento de Datos
   1. Log de la Carga de Datos
   2. Log de Preprocesamiento de Datos
3. Entrenamiento del Modelo
   1. Log del Entrenamiento
   2. Log de Hiperparámetros y Resultados del Modelo
4. Evaluación del Modelo
   1. Log de Evaluación
   2. Log de Resultados

## PREGUNTAS DEL PROFESOR

Como parte de la evaluación del proyecto, cada integrante del equipo deberá demostrar su conocimiento sobre los ejercicios realizados. Esto implica que el profesor realizará **preguntas individuales** para verificar la comprensión de los conceptos, técnicas y resultados obtenidos en el proyecto.

#### **1. Objetivo de las Preguntas**

* Evaluar si todos los integrantes comprenden **todos los aspectos del proyecto** y no solo las partes en las que participaron directamente.
* Garantizar que los integrantes hayan colaborado de manera efectiva y compartidos conocimientos dentro del equipo.
* Identificar posibles áreas de mejora en el entendimiento de los conceptos aplicados.

#### **3. Metodología de las Preguntas**

1. **Preguntas Individuales:** Cada integrante del equipo deberá responder preguntas sobre **cualquier parte del proyecto**, no solo la sección en la que trabajaron directamente.
   * Ejemplo: Aunque un estudiante trabajó principalmente en el preprocesamiento de datos, debe poder explicar los resultados del modelo.
2. **Preguntas Grupales:** En algunos casos, se plantearán preguntas al equipo completo para evaluar cómo integraron las diferentes partes del proyecto y si entienden el flujo general del trabajo.
3. **Demostración Práctica:** El profesor puede pedir a algún integrante que ejecute una parte del código para demostrar cómo funciona y explicar los resultados.

## ENTREGAS

El proyecto debe entregarse en el repositorio de Git en base a la estructura de carpetas definidas previamente. Se subirá el contenido a la rama main que será la rama que se corrija. Únicamente se harán correcciones sobre esa rama. A la hora de realizar la entrega, será necesario crear un TAG en la rama main para determinar que ese es el código que el grupo da por bueno.

Será necesario incluir en el documento del enunciado un apartado en donde se incluya:

* Ruta del repositorio en GitHub
* Identificador de TAG de entrega
* Pantallazos de los avances del proyecto en imágenes del tablos Kanban, donde tendremos definidas las tareas, a quien se ha asignado y en que proceso están. Si el proyecto dura dos semanas, por lo menos habrá que subir 5 pantallazos del estado del tablero Kanban.
  + 1 Estado inicial (Asignadas primeras tareas definidas)
  + 3 Estado intermedio (Debe verse previsión de tiempo, a quien ha sido asignado, y si esta TODO, DOING, DONE)
  + 1 Estado final (Estado del Tablero al crear el TAG)
* Configuración de Loggin: De que forma se ha instalado, configurado y utilizado en el proyecto
* Documentación: Detalle sobre cómo se ha creado la documentación del docstring.

## MODO DE TRABAJO

El proyecto se va a realizar en parejas

### GRUPOS

Los grupos son los siguiente:

|  |  |
| --- | --- |
| **GRUPO 1** |  |
| **GRUPO 2** |  |
| **GRUPO 3** |  |
| **GRUPO 4** |  |
| **GRUPO 5** |  |
| **GRUPO 6** |  |
| **GRUPO 7** |  |
| **GRUPO 8** |  |
| **GRUPO 9** |  |

**FECHA DE INICIO**: 03/12/2024

**FECHA DE ENTREGA**: 14/12/2024 a las 23:00

## EVALUACIÓN

* **Parte 1. Aprendizaje Supervisado** – Análisis de fraude en transacciones con tarjetas de crédito : Peso total 40% de la nota
* **Parte 2. Aprendizaje NO Supervisado** – Clustering de caras con K-Means y PCA: Peso total 40% de la nota
* **Parte 3. Control de versiones, documentación y logging** : Peso total 20% de la nota

### PARTE 1: APRENDIZAJE SUPERVISADO (40%) - Análisis de fraude en transacciones con tarjetas de crédito

En cada apartado cada fallo u omisión resta 0,25

|  |  |  |
| --- | --- | --- |
|  |  | PUNTUACIÓN |
| Explorar los datos | Estadísticos  Gráficos  Matriz de correlaciones  Explicar lo que se aprecia en la matriz de correlaciones | 1,25 |
| Preparar los datos | Dividir los datos en entrenamiento y pruebas  Pipeline que haga todas las transformaciones necesarias  Explicar con palabras las transformaciones que se quieren realizar | 2,50 |
| Entrenar y evaluar en el conjunto de entrenamiento con distintos modelos (también deberías probar algunos ensamblajes) | Entrenar y evaluar en el conjunto de entrenamiento varios modelos (4 + ensamblaje):   * Entrenar con validación cruzada * Decidir la métrica a utilizar y justificar la elección * Elegir justificadamente uno de los modelos | 2,50 |
| Optimizar el mejor modelo del paso anterior | Optimizar el modelo elegido ajustando los 2 o 3 hiperparámetros que consideres más importantes (y explicar que significa cada uno de estos hiperparámetros y que valor puede tomar) | 1,25 |
| Evaluar en el conjunto de pruebas | Evaluar en el conjunto de pruebas el mejor modelo obtenido en el paso anterior. | 0,50 |
| Guardar el modelo | Guardar el modelo en un archivo | 0,10 |
| Presentar de la solución (script aparte). | Crear un script para hacer nuevas predicciones:   * Importar modelo: controlar que el fichero exista. * Importar csv con nuevos datos: pedir la ruta del archivo hasta que el usuario proporciona una ruta correcta. * Predecir si la transacción es un fraude o no. Devolver un dataframe con el número de transacción (nº fila del csv) y el texto Fraude o No fraude. * Controlar todas las posibles excepciones que puedan ocurrir. | 1,90 |
| Comparativa de resultados | Comparativa entre los resultados de todos los grupos, el modelo con mejor resultado en sus métricas, mas ajustado y preciso recibirá un punto extra en total.  Si el proyecto ya tuviera un 10, se añadiría a la nota del examen. | **+1,00** |
|  |  |  |

**NOTA: No se puede dejar ningún apartado en blanco o con puntuación 0, si hay un apartado en blanco la nota de esta parte no será superior a 4**

### PARTE 2: APRENDIZAJE NO SUPERVISADO (40%) - CLUSTERING DE CARAS CON K-MEANS Y PCA

En cada apartado cada fallo u omisión resta 0,25

|  |  |  |
| --- | --- | --- |
|  |  | PUNTUACIÓN |
| Aplica PCA a X para reducir la dimensionalidad | Cargar los datos  Aplicar PCS conservando el 99% de la varianza  Nº de componentes actuales | 1,50 |
| K-Means con k aleatorio | Elegir k  Aplicar k-means  Visualizar grupos  Valorar el resultado (a la vista de las imágenes) | 1,50 |
| K-Means con el mejor k | Utilizar puntaciones de la silueta para elegir k  Aplicar k-means  Visualizar grupos  Valorar el resultado (a la vista de las imágenes) | 2,00 |
| K-Means con el mejor k y sin PCA | Utilizar puntaciones de la silueta para elegir k  Aplicar k-means  Visualizar grupos  Valorar el resultado (a la vista de las imágenes) | 1,50 |
| Comparar los resultados y sacar conclusiones |  | 0,75 |
| Crear el modelo final y guardarlo | Crear modelo  Entrenarlo  Guardar el modelo | 0,75 |
| Script para predecir nuevos dtos | Importar modelo: controlar que el fichero exista  Importar imagen: pedir la ruta del archivo hasta que el usuario proporciona una ruta correcta  Predecir el clúster para la imagen (devolver el número del clúster)  Controlar todas las posibles excepciones que puedan ocurrir | 2,00 |

**NOTA: No se puede dejar ningún apartado en blanco o con puntuación 0, si hay un apartado en blanco la nota de esta parte no será superior a 4**

### PARTE 3: CONTROL DE VERSIONES, DOCUMENTACIÓN Y LOGGING(20%)

En cada apartado cada fallo u omisión resta 0,25

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  | PUNTUACIÓN | |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Estructura de proyecto bien creada y mantenida | | | 1 |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Correcto manejo de ramas y de validaciones | | | 1 |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Fusiones de rama de forma adecuada. Utilización de los elementos Pull Request adecuadamente | | | 1 |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Histórico del proceso de desarrollo correctamente visible en repositorio | | | 1 |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Correcto manejo de la gestión de proyecto Issues y tablero Kanban (Pruebas de la situación del tablero Kanban en diferentes momentos del proyecto) | | | 1 |
| Git / GitHub(Manejo correcto de Git en el proyecto) | Creación de TAG identificador de entrega | | | 1 |
| Comentarios | **Descripción general del código** | | | 0,50 |
| Comentarios | **Comentarios en bloques de código importantes** | | | 0,50 |
| Comentarios/ docstring | **Comentarios en funciones (docstring)** | | | 0,50 |
| docstring | Creación de documentación en HTML (pdoc) | | | 0,50 |
| Logging | Instalación y configuración básica | | | 0,50 |
| Logging | Añadir trazas que documente n el proceso de ejecución de ambas partes y las envíe a el fichero proc\_ejecucion.log | | | 0,75 |
| Logging | Añadir trazas que documente n el proceso de ejecución de ambas partes y las envíe a el fichero errores.log | | | 0,75 |

**NOTA: No se puede dejar ningún apartado en blanco o con puntuación 0, si hay un apartado en blanco la nota de esta parte no será superior a 4**

### PARTE 4: PREGUNTAS PROFESOR

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | |  | PUNTUACIÓN | |
| Preguntas profesor | Preguntas individuales - Cada pregunta no respondida correctamente | | | -0.25 |
| Preguntas profesor | Preguntas grupales | | | -0.25 |
| Preguntas profesor | Demostración practica | | | -0.5 |