Proyecto

Profesor: Fernando Crema Garcia
Fecha: 04 de septiembre de 2024

La nota final será basada el promedio ponderado de cada uno de los componentr4es del proyecto multiplicados por un coeficiente **extra** ($\alpha \in [0{,}8, 1{,}2]$) dependiendo de la calidad del reporte asociado.

Diseño aplicación	API	Notebook	α
20 %	10 %	70 %	[0,8, 1,2]

Fecha de entrega: 14/10/2024 23:59 pm Caracas.
Penalizaciones: 2 ptos / día. Máximo 10 ptos.
Nota final:

$$\text{mín}\left\{\alpha\left(0{,}2D + 0{,}1API + 0{,}7NB\right), 20\right\}$$

I. Introducción

I-A. Objetivos

El objetivo de esta actividad es reforzar las habilidades adquiridas hasta el momento para aprendizaje supervisado y no supervisado creando una aplicación en Streamlit que consuma de un API que tenga almacenado el modelo optimal entrenado por el alumno.

Objetivos específicos

1. Crear un Jupyter Notebook donde explique todo el pipeline de creación del modelo incluyendo: explicación del problema, análisis de los datos, preprocesamiento de datos, selección de modelos y almacenamiento del modelo óptimo, resultados y discusión de resultados. [1]
2. Crear una aplicación en Streamlit que contenga la solución al problema que quieran resolver.
3. Crear un API con FastAPI que sea consumido por la aplicación en la fase de inferencia en caso no supervisado o de predicción en caso supervisado.

II. Evaluación

II-A. Reproducibilidad

Cada uno de los objetivos tiene una ponderación distinta dando prioridad al notebook donde comparan los modelos.

Para lograr que el grupo docente reproduzca sus resultados, deben asignar al comienzo de su entregable la semilla referente a su **cédula de identidad**

II-B. Modelos entrenados

El notebook **por sección** en el cual deben explicar toda la lógica de entrenamiento de los modelos desde preprocesamiento de datos, selección de variables, selección de hiper parámetros y evaluación de modelos. Se deja a juicio del estudiante agregar tanta información sea necesaria para justificar sus hallazgos (gráficos, tabla de resultados, etcétera) asumiendo que el Notebook va a ser corrido en un ambiente independiente [2].

De igual manera, se deben respetar las siguientes restricciones:

1. El Jupyter Notebook debe presentar un uso adecuado de los ítems de headings (#) separando cada nivel dependiendo de la sección involucrada y como es usual se deja a juicio de ustedes la configuración del mismo.
2. Asuman que el Notebook lo va a leer una persona del equipo de toma de decisiones de una empresa pero que tiene conocimiento previo en Aprendizaje Automático [3]. Sean tan detallados como sea posible justificando cada decisión. **No hay soluciones únicas**.
3. Incentivo la creatividad de ustedes y formará parte de la evaluación de α
4. De escoger un dataset de Kaggle **No se pueden usar soluciones que provee la plataforma**. Nunca es un problema revisar soluciones de terceros. Sin embargo, su solución debe ser producto de **su** trabajo. Plagios serán penalizados con nota mínima.
5. Incentivo la colaboración como descrito en el **código de honor** de la materia. Sin embargo, copias serán penalizadas con nota mínima.

III. Jupyter Notebook

Es el entregable **más importante** del proyecto. En él deben explicar todo el proceso de decisiones que tomaron desde presentación del problema hasta el almacenamiento del modelo optimal.

email: fernando.cremagarcia@kuleuven.be

[1] El criterio de optimalidad queda a juicio de ustedes.

[2] Esto significa que no puede depender de archivos externos, si tienen alguna duda de cómo lograr esto contacte al grupo docente

[3] Esto quiere decir que la explicación teórica de los modelos debe ser mínima pero el análisis de resultados es de suma importancia

III-A. *Explicación del problema*

Primero, se debe definir claramente el problema que se desea resolver. Por ejemplo, si se trata de un problema de clasificación, regresión o agrupamiento. Es importante detallar la relevancia del problema y su impacto en el contexto de aplicación. Si es de su campo laboral, explique limitaciones sobre uso de los datos en caso de haber [4].

III-B. *Análisis de los datos*

El análisis de los datos comienza con la carga del dataset y la visualización inicial. Se recomienda usar librerías como `pandas` y `matplotlib` para realizar un análisis exploratorio de los datos (EDA). Algunos pasos importantes incluyen:

- Resumen estadístico de las características.
- Identificación de valores nulos o atípicos.
- Visualización de las distribuciones de las variables y sus correlaciones.

III-C. *Preprocesamiento de los datos*

En esta etapa, se deben realizar los siguientes pasos:

- Limpieza de datos: manejo de valores faltantes, codificación de variables categóricas y normalización o estandarización de los datos.
- División del conjunto de datos en entrenamiento y prueba, utilizando `train_test_split` de `scikit-learn` o de técnicas como validación cruzada.
- Selección de características relevantes si es necesario.

III-D. *Selección de modelos*

El siguiente paso es probar varios algoritmos de aprendizaje supervisado o no supervisado. Ejemplos de modelos incluyen:

- Algoritmos de clasificación como `SVM`, `Árboles de decisión`, `KNN`.
- Algoritmos de agrupamiento como `K-medias` o `agrupamiento jerárquico`.
- Utilizar `cross_val_score` para evaluar el rendimiento mediante validación cruzada.
- Ajuste de hiperparámetros con `GridSearchCV` o `RandomizedSearchCV`.

Una vez identificado el modelo óptimo, se debe almacenar usando `joblib` o `pickle` para poder ser utilizado en producción.

[4]Tomen en cuenta posibles implicaciones éticas. Si no está seguro o segura contacte al grupo docente

III-E. *Resultados y discusión*

Los resultados deben evaluarse con métricas apropiadas. Por ejemplo:

- Para clasificación: `accuracy`, `precision`, `recall`, `F1-score`.
- Para regresión: `MSE`, `RMSE`, `R2`.

Además, es importante visualizar los resultados (por ejemplo, mediante matrices de confusión o curvas ROC) y discutir posibles mejoras al modelo.

IV. Aplicación en Streamlit

La aplicación de Streamlit permitirá a los usuarios interactuar con el modelo de forma sencilla.

IV-A. *Estructura de la aplicación*

- **Introducción**: Breve descripción del problema y cómo interactuar con la aplicación.
- **Entrada de datos**: Uso de widgets como `st.text_input`, `st.number_input` o `st.selectbox` para recolectar la entrada del usuario.
- **Predicción**: La aplicación enviará la entrada del usuario al API para obtener la predicción y mostrar el resultado.

V. API con FastAPI

El API será el puente entre el modelo entrenado y la aplicación de Streamlit.

V-A. *Estructura básica de FastAPI*

- Cargar el modelo entrenado.
- Recibir los datos enviados por la aplicación.
- Devolver la predicción del modelo.

V-B. *Instalación*

Instalar `FastAPI` y `uvicorn` ejecutando el siguiente comando:

```
pip install fastapi uvicorn
```

V-C. *Código básico de FastAPI*

El siguiente código crea un API que recibe datos y devuelve una predicción usando un modelo entrenado:

```
from fastapi import FastAPI
import joblib

app = FastAPI()

model = joblib.load("modelo_entrenado.pkl")

@app.post("/predict")
```

```
async def predict(data: dict):
    input_data = data['data']
    prediction = model.predict([input_data])
    return {"prediction": prediction[0]}
```

V-D. Correr el servidor FastAPI

Para correr el servidor, utiliza `uvicorn`:

```
uvicorn main:app --reload
```

El API estará disponible en `http://localhost:8000/predict`.

VI. Diagrama de Interacción: Streamlit, API y Modelo

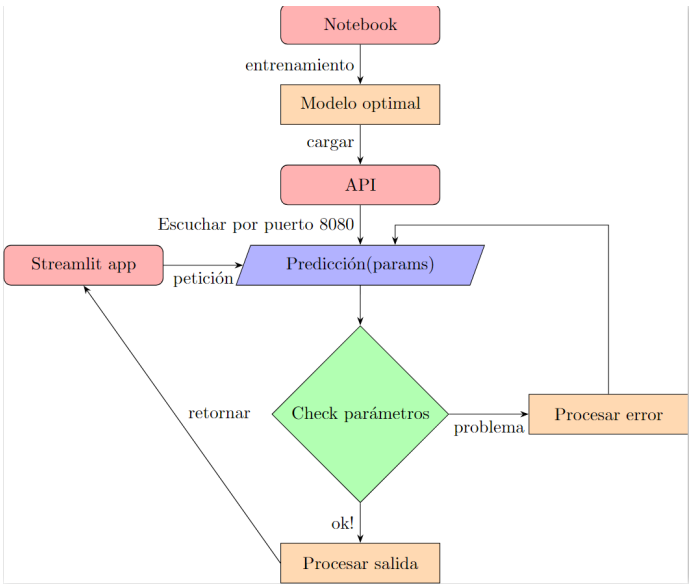

Figura 1. Flujo simplificado de lo mínimo esperado. Comenzamos desde el nodo superior Notebook que luego del proceso de entrenamiento deberá generar el modelo optimal. Luego, el modelo lo debemos almacenar en el mismo sistema de archivos del API que deberá tener como mínimo 1 método de predicción con los parámetros que decidan usar. El modelo de predicción puede ser ejecutado solamente desde la aplicación de streamlit luego de hacer una petición HTTP en el puerto indicado por ustedes. Al llegar la petición, deben revisar que los parámetros son adecuados para generar la salidad que luego será procesada y retornada a la aplicación. En el caso de que ocurra un problema, debe procesar el error y regresar al método de predicción que deberá retornar una estructura que la aplicación entienda ha sido generada luego de un error.