Universidad Nacional de General Sarmiento

Materia: Proyecto Profesional I

Trabajo Práctico: TP Inicial – Documentación e instrucciones de ejecución.

Integrantes: Nereo Manganiello, Nazareno Sotomayor

Docentes: Evelin Aragón, Juan Carlos Monteros

Descripción del modelo

En este trabajo, nos enfocamos en la opción 5, Predicción de riesgo de salud ocupacional. Para entrenar y hacer predicciones con nuestra IA, utilizamos técnicas de machine learning, específicamente las de regresión logística y árboles de decisión, con el objetivo de identificar empleados en “alto riesgo” o “bajo riesgo”, basándose en las horas trabajadas semanales, ausencias, la edad y el salario.

Algoritmos de clasificación

En un principio, nuestro prototipo se basaba en procesar y organizar la información, utilizamos una librería llamada Pandas, más precisamente sus dataframes. El archivo *CSV* que generamos tiene las columnas: ID, Nombre, Edad, *Salario,* Horas_Trabajadas, Ausencias, Genero. Separamos las columnas ID, Edad, Salario, Horas Trabajadas y Ausencias en un mismo dataframe, llamémosle “A”, por otro lado tenemos un *dataframe* que contiene ID, Nombre y Riesgos, llamémosle “B”. Para generar la columna de riesgos calculamos una probabilidad tomando los datos de las horas trabajadas sobre el máximo de las horas trabajadas, ausencia sobre el máximo valor de las ausencias, la edad sobre el máximo valor de la edad y el salario sobre el máximo valor de los salarios, y si la probabilidad es mayor a 0.6 entonces se añadirá en la columna de riesgo un 1 (uno), en caso contrario un 0 (cero). Esto lo hacemos ya que para realizar el entrenamiento necesitamos la columna binaria con la que trabajan los algoritmos de clasificación, de modo tal que el modelo pueda “aprender” de esta columna cuando un empleado es de “alto riesgo” o no.
Separarlos de esta forma nos permitió manejar los datos de manera estructurada. También aplicamos una técnica llamada *One-Hot Encoding* para convertir ciertas categorías en valores numéricos y facilitar su análisis. El modelo que usamos para la predicción fue un árbol de decisión, el cual entrenamos utilizando el 20% de los datos totales del CSV, luego hicimos un scaler para la variable x (la cual toma los valores del dataframe “A”), haciendo que nos queden todos los datos en una misma variable y no separada en dos, como por ejemplo *x_test* y *x_train*. Esto nos permitió medir su rendimiento y asegurarnos de que las predicciones fueran lo más acertadas posible, realizando las mismas en todo el conjunto de datos y no sobre una proporción del mismo. Para evaluar qué tan bien funcionaba el modelo, utilizamos distintas métricas (*precisión*, *memoria* y *F1-score*) y creamos gráficos que nos ayudaron a visualizar los resultados (*confussion matrix* y curva *ROC*). Uno de estos gráficos fue una matriz de confusión, que nos permitió ver cuántas veces el modelo acertó y cuántas se equivocó en sus predicciones. También implementamos el modelo con el algoritmo de regresión logística, funciona exactamente de la misma manera la preparación de los datos en los *dataframes* y el *scaler* (esto sucede porque ambos son algoritmos de clasificación), la diferencia es la manera en que llamamos a la función predefinida del algoritmo a utilizar, la cual en el caso de árboles de decisión lo hacemos con *DecisionTreeClassifier* y en el caso de la regresión logística con *LogisticRegression.*

Features

Además de tener los algoritmos, contamos con distintas funciones que nos ayudan a integrar las *features* que hemos agregado, por ejemplo, contamos con la posibilidad de generar el archivo csv, para eso tenemos la función *generar_csv*, la cual genera valores aleatorios y los agrega a las filas del archivo generando así empleados. Para poder utilizar este archivo, contamos con una función para subir el mismo que se ha generado y guardado en nuestra PC u optar por subir un archivo propio que se quiera evaluar.
También contamos con funciones de verificación, como *verificar_columnas*, que se encarga de verificar que estén todas las columnas requeridas en el csv cargado.
Para las imágenes, contamos con funciones específicas que reciben los parámetros requeridos de los modelos y con ellos generan las imágenes y las guardan dentro de *static/imagenes*, para esto utilizamos la librería *matplotlib* y métricas incluidas en *sklearn* como la curva de **aprendizaje**.
Contamos con una sección especial para visualizar cada empleado clasificado como “alto riesgo”, en esta sección podremos hacer click en cada empleado y visualizar sus características.
Además de tener la posibilidad de generar un archivo con empleados y subirlo para hacer las predicciones, podemos ingresar manualmente un empleado y que el modelo realice la predicción. Para esto se genera un archivo con 500 empleados para entrenar el modelo con un 20% de estos y posteriormente, gracias a los datos del entrenamiento del modelo realizado con el archivo generado, poder hacer la predicción del empleado que hemos ingresado manualmente. De esta forma, podemos predecir individualmente el riesgo de salud de un empleado en particular.
Por último, se añadió un sistema básico de historial de usuarios. Cada vez que una persona accede a la aplicación, debe ingresar un nombre de usuario. Este mecanismo, aunque no tiene medidas de seguridad como contraseñas, sirve para llevar un registro organizado de los usuarios que interactúan con el sistema. Esta información se guarda en un archivo llamado “historial.json” , donde se almacena el nombre del usuario y si subió o no un archivo CSV. Desde el menú "Historial", es posible visualizar esta información en una tabla y, si se desea, con un botón es posible limpiar el historial para comenzar nuevamente desde cero.

Interfaz y Flask

Luego, exploramos algunas herramientas para poder visualizar estos datos a través de una interfaz. En un principio, utilizamos la librería *Streamlit*, pero finalmente prescindimos de esta, ya que consideramos que era más fructífero para nuestro proyecto en general utilizar Flask y combinarlo con *HTML*, *JavaScript* y *CSS* para lograr una interfaz que pudiera satisfacer nuestras necesidades y la de un hipotético usuario, siendo fácil e intuitivo interactuar con la misma. Estas tecnologías nos permitieron crear mayor interactividad, ya que tuvimos la posibilidad de integrar todas las *features* antes mencionadas, para que el

usuario tenga una mejor experiencia utilizando el algoritmo. Flask es un framework minimalista y fácil de integrar para python, este se ejecuta desde un archivo llamado *app.py*, el cual funciona como una especie de "*main*", en donde podremos tener las funciones necesarias para hacer funcionar nuestro back y conectarlo con el front.

Aparte de esto, mostramos las métricas más importantes de los algoritmos, tales como la *precisión*, la *memoria* y el *F1-score*.

Todo el front fue implementado con HTML, CSS y JavaScript, permitiéndonos generar una interfaz simpática y minimalista, y lo más importante, fácil de utilizar. Cuidando nuestro enfoque en cubrir las posibles necesidades de un hipotético usuario, cada menú cuenta con un botón informativo que explica su funcionalidad y cómo utilizarlo. Nuestro objetivo principal fue lograr que el uso de la aplicación fuera didáctico, accesible y visualmente atractivo.

Tenemos dos interfaces principales para el entrenamiento y la predicción, la de árboles de decisión y la de regresión logística. Cada una de estas ejecuta su respectivo algoritmo.

Requisitos

Para implementar el back de esta "aplicación", como ya mencionamos, utilizamos python.
Para los modelos de machine learning, la librería scikit-learn. Para ejecutar los modelos, necesitaremos contar con esta librería instalada, más concretamente en su versión 1.5.2.
Para conectar el back-end con el front-end utilizamos un framework simple para python, recomendado para aplicaciones pequeñas-medianas llamado Flask.
Si ejecutamos la aplicación por medio de la página donde fue desplegada, no necesitaremos contar con estos requisitos instalados en nuestra PC, en caso contrario, sí.
Luego, para desarrollar la interfaz del front-end, como Flask es un framework web para python, utilizamos HTML, CSS y JavaScript, todos los elementos de la interfaz fueron creados con elementos básicos de HTML, estilizado con CSS y utilizando iconos obtenidos desde Google fonts (botones).

Instrucciones para la ejecución

El código se encuentra en el repositorio en la rama main. También, desplegamos el código en un servidor accesible por el navegador, la aplicación se encuentra en: https://tp1-manganiello-sotomayor.up.railway.app/.

Otra forma que brindamos para ejecutar la aplicación es por medio de una instalación. Dentro del repositorio se encuentra un instalador .bat el cual instala todas las dependencias necesarias para correr la aplicación en un local host, al haberlo implementado con Flask en web, necesitamos tener Flask instalado para correrlo (no hace falta si se usa desde el link que bridamos).
También se puede ejecutar instalando la librería Flask utilizando ***pip install flask*** y luego ejecutando el archivo ***app.py,*** pero si se ejecuta el instalador no es necesario.
Para ambos casos (menos el del servidor), deberemos tener python y pip agregados al PATH del sistema.

Pasos:

1. Abrir la página https://tp1-manganiello-sotomayor.up.railway.app/ o ejecutar *install.bat* (no lo recomendamos ya que la aplicación está desplegada en web).
    1.1.1. Si se instala la aplicación con *install.bat*, esperar a que se instalen las dependencias, la aplicación se ejecutará sola luego de que terminen de instalarse las dependencias, si no lo hace, buscar en el navegador http://127.0.0.1:8080 o http://192.168.0.8:8080. **Importante no cerrar el cmd que se abre para la instalación**.
    1.1.2. Luego que termine, se abrirá sola la interfaz web en el navegador.
2. Se abrirá la pantalla de inicio, la cual es un menú para navegar hacia las diferentes funcionalidades de la aplicación.
3. Si elegimos predecir, iremos a una pantalla la cual podremos generar el archivo csv (**una vez apretado el botón para generar, se descargará**) con la cantidad de empleados que pasemos por parámetro.
    3.1. Tendremos que cargar el archivo que se guardará en la carpeta indicada por la interfaz luego de ser generado. Luego podremos apretar el botón de entrenar y posteriormente el de predecir. Una vez hecho esto, se nos mostrará las métricas del algoritmo y podremos descargar el CSV con las predicciones, ver todos los empleados clasificados como “alto riesgo” y por último visualizar los gráficos generados.
4. Si elegimos predecir individual, tendremos que ingresar manualmente los datos del empleado: Horas trabajadas, ausencias, edad y salario. Luego de introducir todos estos parámetros podremos predecir el empleado, para posteriormente visualizar los datos en la interfaz.

Cómo entrenarlo y hacer predicciones

Como mencionamos en los pasos para la ejecución, existen dos modalidades principales de predicción:

1. **Predicción con archivo CSV**: Tendremos dos opciones para entrenar el modelo y predecirlo, árboles de decisión y regresión logística, cada uno con su propia interfaz. Para ambos modelos, el usuario puede generar un archivo de empleados (o subir uno propio) y luego cargarlo desde la interfaz. Posteriormente, debe presionar el botón de entrenamiento para que el modelo aprenda a partir de esos datos, y luego hacer clic en "Predecir" para obtener los resultados.
2. **Predicción individual**: En esta modalidad, el usuario ingresa manualmente los datos de un solo empleado a través de un formulario. El sistema, en segundo plano, entrena el modelo automáticamente con un conjunto de 500 empleados generados aleatoriamente, y luego realiza la predicción utilizando los datos recién ingresados.

Conclusión

En conclusión, para este trabajo optamos por desarrollar la opción 5: Predicción de riesgo de salud ocupacional. Aplicamos técnicas de *machine learning* como regresión logística y árboles de decisión para abordar la problemática planteada y ofrecer una solución efectiva. Además, construimos una interfaz interactiva, intuitiva y visualmente agradable, que facilita la interacción con el sistema, incluso para usuarios sin conocimientos técnicos. Creemos que nuestro proyecto cumple con los requisitos establecidos y nos sentimos satisfechos con los resultados obtenidos. Sin embargo, reconocemos que aún existen aspectos por mejorar, y consideramos este desarrollo como una base sólida sobre la cual continuar aprendiendo y mejorando en futuros desafíos.