FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

PROYECTO INTELIGENCIA ARTIFICIAL

TITULO	Capstone Project-IBM Employee Attrition Prediction
AUTOR	**Nombre y Apellido**
	Churqui Mamani Angel
FECHA	20/05/2023

Carrera	Informática Asignatura
Asignatura	INF - 354
Docente	Lic. Moises Silva

RESUMEN

"IBM Employee Attrition Prediction" es un proyecto que utiliza técnicas de aprendizaje automático para predecir la atrición de empleados en una organización. El objetivo es desarrollar un modelo de clasificación que pueda identificar qué empleados son propensos a abandonar la organización. El proyecto implica el análisis de datos proporcionados por IBM, la limpieza y preprocesamiento de los datos, la selección de características relevantes y la construcción de un clasificador de árbol de decisión. El modelo entrenado se evalúa utilizando métricas como la precisión y la matriz de confusión. El objetivo final es ayudar a la organización a tomar medidas proactivas para retener a los empleados clave y reducir la rotación de personal.

El análisis de preprocesamiento en el proyecto "IBM Employee Attrition Prediction" se refiere a las etapas de limpieza y transformación de los datos antes de ser utilizados para entrenar el modelo de predicción de atrición de empleados.

En general, el análisis de preprocesamiento se centra en garantizar la calidad de los datos, convertir datos no numéricos a numéricos cuando sea necesario y dividir los datos en conjuntos adecuados para el entrenamiento y la evaluación del modelo. Estas etapas son fundamentales para garantizar resultados precisos y confiables en el modelo de predicción de atrición de empleados.

Para la parte de procesamiento se utiliza el siguiente código esto para rellenar los datos que esten en blanco:

```
import pandas as pd
import numpy as np
import seaborn as sns

#Abrimos el archivo
df = pd.read_csv('/content/drive/MyDrive/dataset/IBM HR.csv')

#Preprocesamiento de datos
#Rellenar los datos faltantes (Categóricos)
modas = df[['Attrition',
          'BusinessTravel',
          'Department',
          'Education',
          'EducationField',
          'EnvironmentSatisfaction',
          'Gender',
          'JobInvolvement',
          'JobLevel',
          'JobRole',
          'JobSatisfaction',
          'MaritalStatus',
          'OverTime',
          'PerformanceRating',
          'RelationshipSatisfaction',
```

```
          'WorkLifeBalance']].mode()

df.loc[pd.isna(df["Attrition"]),'Attrition']=modas['Attrition']
df.loc[pd.isna(df["BusinessTravel"]),'BusinessTravel']=modas['Business
Travel']
df.loc[pd.isna(df["Department"]),'Department']=modas['Department']
df.loc[pd.isna(df["Education"]),'Education']=modas['Education']
df.loc[pd.isna(df["EducationField"]),'EducationField']=modas['Educatio
nField']
df.loc[pd.isna(df["EnvironmentSatisfaction"]),'EnvironmentSatisfaction
']=modas['EnvironmentSatisfaction']
df.loc[pd.isna(df["Gender"]),'Gender']=modas['Gender']
df.loc[pd.isna(df["JobInvolvement"]),'JobInvolvement']=modas['JobInvol
vement']
df.loc[pd.isna(df["JobLevel"]),'JobLevel']=modas['JobLevel']
df.loc[pd.isna(df["JobRole"]),'JobRole']=modas['JobRole']
df.loc[pd.isna(df["JobSatisfaction"]),'JobSatisfaction']=modas['JobSat
isfaction']
df.loc[pd.isna(df["MaritalStatus"]),'MaritalStatus']=modas['MaritalSta
tus']
df.loc[pd.isna(df["OverTime"]),'OverTime']=modas['OverTime']
df.loc[pd.isna(df["PerformanceRating"]),'PerformanceRating']=modas['Pe
rformanceRating']
df.loc[pd.isna(df["RelationshipSatisfaction"]),'RelationshipSatisfacti
on']=modas['RelationshipSatisfaction']
df.loc[pd.isna(df["WorkLifeBalance"]),'WorkLifeBalance']=modas['WorkLi
feBalance']

#Rellenar los datos faltantes (Numéricos)
medias = df[['Age',
             'DistanceFromHome',
             'MonthlyIncome',
             'NumCompaniesWorked',
             'PercentSalaryHike',
             'TotalWorkingYears',
             'TrainingTimesLastYear',
             'YearsAtCompany',
             'YearsInCurrentRole',
             'YearsSinceLastPromotion',
             'YearsWithCurrManager']].mean()
print(df)
df.loc[pd.isna(df["Age"]),'Age']=medias['Age']
df.loc[pd.isna(df["DistanceFromHome"]),'DistanceFromHome']=medias['Dis
tanceFromHome']
```

```
df.loc[pd.isna(df["MonthlyIncome"]),'MonthlyIncome']=medias['MonthlyIn
come']
df.loc[pd.isna(df["NumCompaniesWorked"]),'NumCompaniesWorked']=medias[
'NumCompaniesWorked']
df.loc[pd.isna(df["PercentSalaryHike"]),'PercentSalaryHike']=medias['P
ercentSalaryHike']
df.loc[pd.isna(df["TotalWorkingYears"]),'TotalWorkingYears']=medias['T
otalWorkingYears']
df.loc[pd.isna(df["TrainingTimesLastYear"]),'TrainingTimesLastYear']=m
edias['TrainingTimesLastYear']
df.loc[pd.isna(df["YearsAtCompany"]),'YearsAtCompany']=medias['YearsAt
Company']
df.loc[pd.isna(df["YearsInCurrentRole"]),'YearsInCurrentRole']=medias[
'YearsInCurrentRole']
df.loc[pd.isna(df["YearsSinceLastPromotion"]),'YearsSinceLastPromotion
']=medias['YearsSinceLastPromotion']
df.loc[pd.isna(df["YearsWithCurrManager"]),'YearsWithCurrManager']=med
ias['YearsWithCurrManager']
print(df)

#Reemplazar datos categóricos con valores numéricos
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

df["Attrition"] = encoder.fit_transform(df.Attrition.values)
df["BusinessTravel"] = encoder.fit_transform(df.BusinessTravel.values)
df["Department"] = encoder.fit_transform(df.Department.values)
df["EducationField"] = encoder.fit_transform(df.EducationField.values)
df["Gender"] = encoder.fit_transform(df.Gender.values)
df["JobRole"] = encoder.fit_transform(df.JobRole.values)
df["MaritalStatus"] = encoder.fit_transform(df.MaritalStatus.values)
df["OverTime"] = encoder.fit_transform(df.OverTime.values)

#Estandarizacion
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['MonthlyIncome']] =
scaler.fit_transform(df[['MonthlyIncome']])
```

por otro lado para lograr lo que es la clasificacion ocupando el arbol de decision se hizo lo siguiente.

CLASIFICACION USANDO EL ÁRBOL DE DECISION

Clasificación utilizando el árbol de decisión es una técnica común en el análisis de datos y aprendizaje automático. El árbol de decisión es un algoritmo de aprendizaje supervisado que construye un modelo de predicción en forma de estructura de árbol. Se utiliza para realizar clasificaciones basadas en la evaluación de características del conjunto de datos. Aquí hay una descripción general de cómo realizar clasificación utilizando el árbol de decisión: Preparación de los datos: Antes de construir el árbol de decisión, es necesario preparar los datos. Esto incluye la limpieza de los datos, la selección de características relevantes y la división del conjunto de datos en conjuntos de entrenamiento y prueba.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from imblearn.over_sampling import RandomOverSampler

# Cargar el dataset
data = pd.read_csv('/content/drive/MyDrive/dataset/IBM HR Data
new.csv')

# Preprocesamiento de datos
def data_clean(data):
    # Aquí puedes realizar el preprocesamiento necesario para tu
conjunto de datos
    # Asegúrate de eliminar columnas no relevantes o que contengan
datos no numéricos
    # y realizar cualquier otra transformación necesaria

    return data

data = data_clean(data)
X = data.drop(['Attrition'], axis=1)
y = data['Attrition']

# Aplicar muestreo excesivo para equilibrar las clases
oversampler = RandomOverSampler()
X_resampled, y_resampled = oversampler.fit_resample(X, y)

# Escalar características
scaler = StandardScaler()
X_resampled = scaler.fit_transform(X_resampled)

# Dividir los datos en conjuntos de entrenamiento y prueba
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2, random_state=0)

# Construir el clasificador de árbol de decisión
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Calcular la precisión en los conjuntos de entrenamiento y prueba
train_accuracy = model.score(X_train, y_train)
test_accuracy = model.score(X_test, y_test)
print("Precisión en el conjunto de entrenamiento:", train_accuracy)
print("Precisión en el conjunto de prueba:", test_accuracy)

# Calcular la matriz de confusión
y_pred = model.predict(X_test)
confusion = confusion_matrix(y_test, y_pred)
print("Matriz de Confusión:")
print(confusion)

# Calcular la precisión del clasificador
accuracy = accuracy_score(y_test, y_pred)
print("Precisión del Clasificador:", accuracy)
```