SENATI

Apellidos y Nombres: Huamani Feria Claudio

ID: 001597435

Dirección Zonal/CFP: Arequipa/Arequipa/Puno

Carrera: Ingeniería de software con Inteligencia artificial

Semestre: 4to Semestre

Instructor: Jesús Romero Villanueva

Curso: seminario de complementación práctica I

TAREAS Y OPERACIONES COVERTURADAS

-TH1_Realiza operaciones con las Librerías Pandas y Numpy

- Estudia los fundamentos de vectores y matrices.
- Define la librería Pandas y Numpy.
- Manipula y analiza estructuras de datos.
- Lee archivos CSV con Numpy y Pandas.

-TH2_Estudia el uso de las Librerías Scikit-learn y Pytorch

- Define el concepto de Machine Learning
- Define la librería Scikit-Learn y Pytorch.
- Identifica principales aplicaciones.

-TH3_Estudia el uso de las librerías SciPy y Nltk

- Define que el procesamiento de lenguaje natural (NLP)
- Define la librería SciPy y Nltk.
- Identifica principales aplicaciones del NLP

-TH4_Estudia el uso de las Librerías Tensorflow y Keras

- Define el concepto de Deep Learning
- Define la librería Tensorflow y Keras.
- Identifica principales aplicaciones

-TH5_Realiza operaciones con las librerías Matplotlib y Seaborn

- Define los histogramas y la importancia de la visualización de datos.
- Define la librería Matplotlib y Seaborn.
- Crea ejemplos de aplicación

-TH6_Crea programas con algoritmos de aprendizaje supervisado

- Describe los tipos de algoritmos del aprendizaje supervisado.
- Define la regresión lineal simple y múltiple.

TH6_Crea programas con algoritmos de aprendizaje supervisado

Tarea1:

Se importan los modelos necesarios y numpy. Luego, se crea un arreglo x con los valores [18, 10, 6] y un arreglo y con etiquetas basadas en las condiciones de x (0, 1, 2). Se entrena un modelo de árbol de decisión usando x como entrada y y como salida. Posteriormente, se hacen predicciones para los valores [20, 5, 11]. Finalmente, se imprimen las predicciones resultantes.

```
from sklearn.tree import DecisionTreeClassifier #Multiclase
from sklearn.linear_model import LogisticRegression #Binaria
from sklearn.linear_model import LinearRegression #Regresión lineal
import numpy as np

x=np.array([18,10,6])
y=np.array([0 if i < 18 else 1 if i < 10 else 2 for i in x])
model=DecisionTreeClassifier()
model.fit(x.reshape(-1, 1), y)
y_pred=model.predict(np.array([[20],[5],[11]]))
print(y_pred)
```

Tarea2:

Se importan LinearRegression y numpy. Se crean los datos de entrada X con valores del 1 al 5 y las etiquetas y con valores correspondientes. Se inicializa y entrena un modelo de regresión lineal con esos datos. Luego, se usa el modelo para predecir el valor asociado a la entrada 6. Finalmente, se imprime la predicción obtenida.

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])
y = np.array([30, 35, 40, 45, 50])

modelo_lineal = LinearRegression()
modelo_lineal.fit(X, y)

prediccion = modelo_lineal.predict([[6]])
print(prediccion)
```

Tarea3:

Se importan LogisticRegression y numpy. Se definen los datos de entrada X con valores del 1 al 7 y las etiquetas binarias y (0 o 1). Se crea y entrena un modelo de regresión logística con esos datos. Después, se predice la clase para el valor 3.5 usando el modelo entrenado. Finalmente, se imprime la predicción.

```
from sklearn.linear_model import LogisticRegression
import numpy as np

X = np.array([[1], [2], [3], [4], [5], [6], [7]])
y = np.array([0, 0, 0, 1, 1, 1, 1])

modelo_logistico = LogisticRegression()
modelo_logistico.fit(X, y)

prediccion = modelo_logistico.predict([[3.5]])
print(prediccion)
```

Tarea4:

Se importan DecisionTreeClassifier y numpy. Se definen los datos de entrada x con tres valores y sus etiquetas y correspondientes (0, 1, 2). Se crea y entrena un modelo de árbol de decisión con esos datos. Luego, se predice la clase para el valor 2. Finalmente, se imprime la predicción.

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np

x = np.array([[18], [10], [6]])
y = np.array([0,1,2])

model = DecisionTreeClassifier()
model.fit(x, y)

y_pred = model.predict(np.array([[2]]))
print(y_pred)
```

Tarea5:

Se importan las herramientas necesarias para crear una red neuronal, cargar

datos, dividir el dataset y evaluar el modelo.
Se carga el dataset Iris y se separan las características y las etiquetas.
Se divide el dataset en conjuntos de entrenamiento (80%) y prueba (20%).
Se crea un modelo de red neuronal con dos capas ocultas de tamaños 10 y 8, usando activación ReLU y el optimizador Adam, con un máximo de 1000 iteraciones.
Se entrena el modelo con los datos de entrenamiento.
Se hacen predicciones usando el conjunto de prueba.
Se calcula la exactitud del modelo y se genera un reporte detallado de clasificación.
Finalmente, se imprimen la exactitud, el reporte de clasificación, las etiquetas reales y las predicciones obtenidas.

```python
#importamos MLPClassifier de sklearn para crear una red neuronal
#importamos también algunas utilidades para evaluar el modelo como accuracy_score y classification_report
#importamos train_test_split para dividir el dataset en conjunto de entrenamiento y prueba
#importamos load_iris para cargar el dataset de iris
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

#cargamos el dataset de iris
iris = load_iris()
x = iris.data
y = iris.target

#dividimos el dataset en conjunto de entrenamiento y prueba
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

#el tamaño de la capa oculta es (10, 8) lo hacemos con hidden_layer_sizes
#usamos max_iter para definir el número máximo de iteraciones
model = MLPClassifier(hidden_layer_sizes=(10, 8), activation='relu', solver='adam', max_iter=1000, random_state=42)

#entrenamos el modelo
model.fit(x_train, y_train)

#hacemos predicciones
y_pred = model.predict(x_test)

#evaluamos el modelo
accuracy = accuracy_score(y_test, y_pred)

#hacemos un reporte de clasificación
report = classification_report(y_test, y_pred)

#imprimimos los resultados de la evaluación
print(f"Accuracy: {accuracy}")

#imprimimos la matriz de confusión
print("Classification Report:")
print(report)

#imprimimos las etiquetas reales y las predicciones
print ("Etiquetas reales:", y_test)
print("Predicciones:", y_pred)
```

Tarea 6:

Este código toma datos de casas con características numéricas y categóricas, las **normaliza** y convierte categorías en variables binarias (one-hot). Luego, entrena un modelo **Lasso** para predecir el precio usando esas características procesadas. Finalmente, imprime las predicciones, coeficientes del modelo y métricas de desempeño (R^2 y error cuadrático).

```python
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso, LinearRegression
# One-hot encoding para variables categóricas y StandardScaler para variables numéricas
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
data = pd.DataFrame(
    {
        "tamanio": [50, 80, 60, 120, 100], #StandardScaler
        "habitaciones": [2, 3, 2, 4, 3], #StandardScaler
        "zona": ["Norte", "Sur", "Este", "Oeste"], #OneHotEncoder
        "antiguedad": [10, 5, 20, 2, 15],   #StandardScaler
        "precio": [200000, 300000, 250000, 400000, 350000] #Target
    }
)
x = data[["tamanio", "habitaciones", "zona", "antiguedad"]]
y = data["precio"]
scaler = StandardScaler()
x_num = scaler.fit_transform(data[["tamanio", "habitaciones", "antiguedad"]])
encoder = OneHotEncoder(drop='first', sparse_output=False)
x_cat = encoder.fit_transform(data[["zona"]])
print(x_num)
print(x_cat)

x_final = np.hstack((x_num, x_cat))
print(x_final)

model = Lasso(alpha=0.1)
model.fit(x_final, y)

y_pred = model.predict(x_final)
print("Predicciones: ", y_pred)
print("Coeficientes: ", model.coef_)
print("Intercepto: ", model.intercept_)
print("R^2 Score: ", r2_score(y, y_pred))
print("Mean Squared Error: ", mean_squared_error(y, y_pred))
```

Tarea 7:

El código carga el conjunto de datos de viviendas en California, que contiene características como la ubicación, el tamaño y otros factores, junto con los precios de las casas. Luego divide estos datos en dos grupos: uno para entrenar el modelo y otro para probar su desempeño. Utiliza un modelo de regresión lineal para aprender la relación entre las características y los precios. Después de entrenar, hace predicciones sobre los datos de prueba y evalúa qué tan preciso es el modelo calculando el error cuadrático medio y el coeficiente de determinación (R^2). Finalmente, muestra estos resultados y la descripción completa del conjunto de datos para entender mejor su contenido.

```python
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
california = fetch_california_housing()
x = california.data
y = california.target
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
model = LinearRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred)
print("Mean Squared Error:",mse)
print("R^2 Score:",r2)
print(california.DESCR)
```