CS19411-PYTHON PROGRAMMING FOR MACHINE LEARNING

AKSHAYA HARINI BR

220901004

III YEAR -EEE 'A'

MINI PROJECT REPORT

SMART IRRIGATION SYSTEM USING MACHINE LEARNING

AIM:

This project focuses on developing a Smart Irrigation System leveraging machine learning techniques to optimize water usage in agriculture. By analysing environmental factors such as soil moisture, temperature, humidity, and rainfall predictions, the system ensures efficient water distribution. The model achieved X% accuracy in predicting irrigation requirements, showcasing its potential for sustainable agricultural practices.

SVM Model Implementation:

Step 1: Initiating the hyperparameters .

Step 2: Fitting the dataset to SVM Classifier.

Step 3: Implementing Gradient Descent algorithm for Optimization.

Step 4: Predict the label for a given input value.

PROGRAM CODE:

```
import numpy as np

import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

#  INITIATING THE HYPERPARAMETERS

class SVM:

def __init__(self, learning_rate, no_of_iterations, lambda_parameter):

self.learning_rate = learning_rate

self.no_of_iterations = no_of_iterations

self.lambda_parameter = lambda_parameter


# fitting the dataset to SVM Classifier

def fit(self, X, Y):

# m  --> number of Data points --> number of rows

# n  --> number of input features --> number of columns

self.m, self.n = X.shape

# initiating the weight value and bias value

self.w = np.zeros(self.n)

self.b = 0

self.X = X

self.Y = Y
```

```
# implementing Gradient Descent algorithm for Optimization
for i in range(self.no_of_iterations): # Now within the fit method
self.update_weights()
# function for updating the weight and bias value
def update_weights(self):
# label encoding
y_label = np.where(self.Y <= 0, -1, 1)
# gradients ( dw, db)
for index, x_i in enumerate(self.X): #Fixed: Corrected indentation for this line
condition = y_label[index] * (np.dot(x_i, self.w) - self.b) >= 1
if (condition == True):
dw = 2 * self.lambda_parameter * self.w
db = 0
else:
dw = 2 * self.lambda_parameter * self.w - np.dot(x_i, y_label[index])
db = y_label[index]
self.w = self.w - self.learning_rate * dw
self.b = self.b - self.learning_rate * db
# predict the label for a given input value
def predict(self, X):
output = np.dot(X, self.w) - self.b
predicted_labels = np.sign(output)
y_hat = np.where(predicted_labels <= -1, 0, 1)
return y_hat
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# LOADING THE DATA FROM CSV FILES TO PANDA DATAFRAME
crop_data = pd.read_csv('/content/projectdata2.csv')
#PRINT FIRST 5 ROWS OF THE DATAFRAME
crop_data.head()
#NUMBER OF ROW AND COLUMNS IN A DATASET
crop_data.shape
#GETTING STATISTICAL MEASURES OF THE DATASET
crop_data.describe()
#COUNT OF THE DATASET
if 'Days_to_Harvest' not in crop_data.columns:
print(f"Available columns: {crop_data.columns.tolist()}")
```

```
else:
print(crop_data['Days_to_Harvest'].value_counts())
features = crop_data.drop(columns='irrigation_frequency', axis=1)
target = crop_data['irrigation_frequency']
print(target)
# Import necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
# Assuming 'features' is the DataFrame
# Select only numerical features for scaling
numerical_features = features.select_dtypes(include=['number']).columns
features_to_scale = features[numerical_features]
# Now fit the scaler on the numerical features only
scaler = StandardScaler()
scaler.fit(features_to_scale)
# Transform only the numerical features
standardized_data_numerical = scaler.transform(features_to_scale)
# 1. Create a new DataFrame with standardized numerical features
standardized_data = pd.DataFrame(standardized_data_numerical, columns=numerical_features, index=features.index)
# 2. Concatenate the standardized numerical features with the original non-numerical features
standardized_data = pd.concat([standardized_data, features.drop(columns=numerical_features)], axis=1)
print(standardized_data)
features = standardized_data
target = crop_data['irrigation_frequency']
print(features)
print(target)
X_train, X_test, Y_train, Y_test = train_test_split(features, target, test_size=0.2, random_state = 2)
print(features.shape, X_train.shape, X_test.shape)
from sklearn.svm import SVC
classifier = SVC(kernel='linear', C=1/0.01)
!pip install -q pandas scikit-learn
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
categorical_features = ['crop']
```

```
numerical_features = features.drop(columns=categorical_features).columns
# Create a ColumnTransformer to apply OneHotEncoder to categorical features
# and SimpleImputer to handle NaN values
preprocessor = ColumnTransformer(
transformers=[
('num', SimpleImputer(strategy='mean'), numerical_features),  # Impute numerical features
('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features),
])
# Fit and transform the data
features_encoded = preprocessor.fit_transform(features)
# Split the encoded data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(features_encoded, target, test_size=0.2, random_state=2)
# Create and train the SVC classifier
classifier = SVC(kernel='linear', C=1/0.01)
classifier.fit(X_train, Y_train)
print(features_encoded.shape, X_train.shape, X_test.shape)
# accuracy on training data
X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
print('Accuracy score on training data = ', training_data_accuracy)
# accuracy on training data
X_test_prediction = classifier.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print('Accuracy score on test data = ', test_data_accuracy)
# BUILD A PREDICTIVE SYSTEM
# Input sequence from the user
input_sequence = input("Enter the crop: ")
# Create a DataFrame with all expected columns
input_data = pd.DataFrame({
'crop': [input_sequence],
'temperature': [0],
'humidity':[0],
'ph': [0],
'rainfall':[0],
'sunlight_exposure':[0],
'soil_moisture':[0],
'soil_type':[0],
'wind_speed':[0],
```

```
'water_source_type':[0],
'frost_risk':[0],
'water_usage_efficiency':[0]
})
# Preprocess the input data using the same preprocessor used for training
try:
input_encoded = preprocessor.transform(input_data)
# Make prediction using the trained classifier
prediction = classifier.predict(input_encoded)
# Print the predicted irrigation
print("Predicted irrigation:", prediction[0])
except ValueError as e:
print(f"Error during prediction: {e}")
print("Please ensure the input sequence and other features are in the correct format.")

# Distribution of the target variable
plt.figure(figsize=(14, 12))
sns.countplot(x=target)
plt.title('Distribution of Irrigation Prediction')
plt.show()
#Comparison of training and testing accuracy
accuracy_scores = {'Training': training_data_accuracy, 'Testing': test_data_accuracy}
plt.figure(figsize=(14, 12))
plt.bar(accuracy_scores.keys(), accuracy_scores.values(), color=['lightblue', 'pink'])
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy Score')
plt.ylim([0, 1]) # Set y-axis limit to 0-1 for accuracy
plt.show()
```

OUTPUT:

EXAMPLE 1:

Crop : maize

Irrigation_frequency : 2

EXAMPLE 2:

Crop : chickpeas

Irrigation_frequency : 4

CONCLUSION:

Through the analysis of environmental elements such as rainfall, temperature, and soil moisture, the machine learning-based smart irrigation system effectively optimizes water usage in agriculture. The machine learning approach promotes sustainable farming practices by increasing irrigation efficiency, decreasing water waste, and minimizing manual intervention. Even if the experiment shows encouraging results, bigger datasets, more sophisticated models, and IoT integration for real-time monitoring are possible for future enhancements. Addressing water scarcity and guaranteeing effective resource management in contemporary farming are made possible by this technique.