Plant Based Water Needs

A Project Report

Submitted by
Rudransh Mishra

In partial fulfillment for the award of degree
Of
Bachelor of technology

KIET GROUP OF INSTITUTIONS,GHAZIABAD

Introduction:

Plants require different levels of water based on their environmental conditions and characteristics such as sunlight exposure, watering frequency, and soil type. Incorrect watering can lead to overwatering or drought stress, reducing plant health and productivity.

This project aims to build a machine learning model that classifies a plant's water need (`low`, `medium`, or `high`) based on input features: sunlight hours, watering frequency, and soil type.

Methodology:

We followed these steps to build the classification model:

1. **Data Preprocessing:**
    - Loaded and explored the dataset from `plants.csv`.
    - Applied one-hot encoding to the categorical column `soil_type`.
    - Split the dataset into features (`X`) and target (`y`).
2. **Model Selection:**
    - Chose a Decision Tree Classifier due to its interpretability and simplicity.
3. **Training and Evaluation:**
    - Data was split into 80% training and 20% testing sets.
    - Model was trained on the training set.
    - Accuracy and classification report were generated from test predictions.

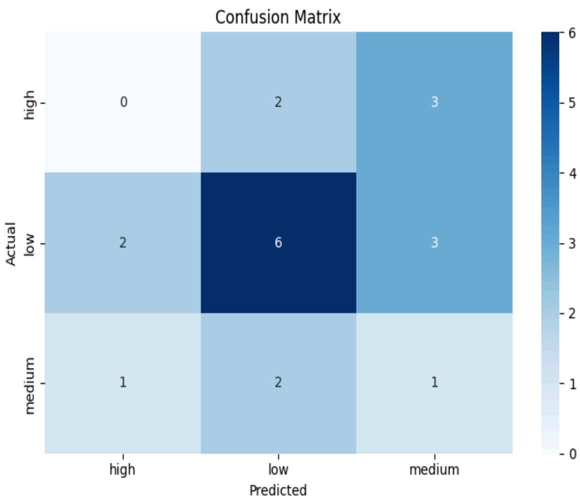
Code:

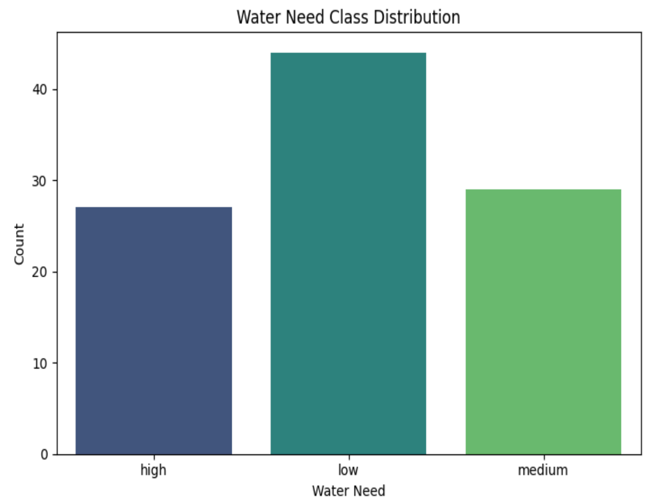
```
import pandas as pd  # Importing pandas for data manipulation
import matplotlib.pyplot as plt  # Importing matplotlib for plotting
import seaborn as sns  # Importing seaborn for enhanced plotting
import warnings  # Importing warnings to manage warning messages
from sklearn.model_selection import train_test_split  # Importing function
to split data into training and testing sets
from sklearn.preprocessing import LabelEncoder  # Importing LabelEncoder
for encoding categorical variables
from sklearn.naive_bayes import GaussianNB  # Importing Gaussian Naive
Bayes classifier
from sklearn.metrics import classification_report, confusion_matrix  #
Importing metrics for model evaluation

# Ignore FutureWarnings to keep the output clean
warnings.simplefilter(action='ignore', category=FutureWarning)

# Load dataset from a CSV file into a DataFrame
df = pd.read_csv("/content/plants.csv")

# Encode categorical columns to numerical values for model compatibility
label_encoders = {}  # Dictionary to store label encoders for each
categorical column
for col in df.columns:  # Iterate through each column in the DataFrame
    if df[col].dtype == 'object':  # Check if the column is categorical
        le = LabelEncoder()  # Create a LabelEncoder instance
        df[col] = le.fit_transform(df[col])  # Fit and transform the
column to numerical values
        label_encoders[col] = le  # Store the encoder for later use

# Define features (X) and target variable (y)
X = df.drop("water_need", axis=1)  # Features: all columns except
'water_need'
y = df["water_need"]  # Target: 'water_need' column

# Split the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```python
# Model training using Gaussian Naive Bayes
model = GaussianNB()  # Create an instance of the Gaussian Naive Bayes
model
model.fit(X_train, y_train)  # Fit the model to the training data

# Predict the target variable for the test set
y_pred = model.predict(X_test)  # Make predictions on the test set

# Generate a classification report to evaluate model performance
report = classification_report(y_test, y_pred, output_dict=True)  # Get
classification metrics
report_df = pd.DataFrame(report).transpose()  # Convert report to
DataFrame for better readability
print(report_df)  # Print the classification report DataFrame

# Create a confusion matrix to visualize prediction results
cm = confusion_matrix(y_test, y_pred)  # Compute confusion matrix
labels = label_encoders["water_need"].classes_  # Get original class
labels for the target variable

plt.figure(figsize=(14, 5))  # Set the figure size for the plots

# Plot Confusion Matrix
plt.subplot(1, 2, 1)  # Create a subplot for the confusion matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels,
yticklabels=labels)  # Create a heatmap for the confusion matrix
plt.title("Confusion Matrix")  # Set the title for the confusion matrix
plot
plt.xlabel("Predicted")  # Label for the x-axis
plt.ylabel("Actual")  # Label for the y-axis

# Class Distribution Bar Chart
plt.subplot(1, 2, 2)  # Create a subplot for the class distribution
class_counts = df["water_need"].value_counts().sort_index()  # Count
occurrences of each class and sort by index
# Use inverse_transform on the full list at once (avoids loop + warning)
inverse_labels =
label_encoders["water_need"].inverse_transform(class_counts.index)  # Get
original labels for the classes
```

```
sns.barplot(x=inverse_labels, y=class_counts.values, palette="viridis")  #
Create a bar plot for class distribution
plt.title("Water Need Class Distribution")  # Set the title for the class
distribution plot
plt.xlabel("Water Need")  # Label for the x-axis
plt.ylabel("Count")  # Label for the y-axis

plt.tight_layout()  # Adjust layout to prevent overlap
plt.show()  # Display the plots
```

Output:


Confusion Matrix
high
low
medium
Actual
0
2
3
2
6
3
1
2
1
high
low
medium
Predicted
6
5
4
3
2
1
0


Water Need Class Distribution
Count
0
10
20
30
40
high
low
medium
Water Need

References:

- **Tools Used**: Python, pandas, scikit-learn
- **Model**: Decision Tree Classifier
- **Platform**: Jupyter Notebook / Python Execution Environment
- **Documentation and Concepts**:
    - Scikit-learn Documentation: https://scikit-learn.org/stable/
    - Pandas Documentation: https://pandas.pydata.org/docs/
    - Python Official Documentation: https://docs.python.org/3/
- **Inspiration**:
    - Decision Tree theory: https://en.wikipedia.org/wiki/Decision_tree_learning
    - Watering needs in horticulture: https://www.gardeningknowhow.com/
-