Predicting Soil Moisture Using Hyperspectral Data

Project Overview

This project aims to predict soil moisture levels using a dataset containing hyperspectral data, soil temperature, and moisture values. The dataset includes various spectral bands and environmental measurements over time. The goal is to preprocess the data, perform exploratory data analysis (EDA), select relevant features, and build a predictive model using machine learning techniques.

1. Data Collection and Preprocessing

The dataset is imported and pre-processed to handle missing values, convert datetime formats, and scale the features for better model performance.

- **Datetime Handling**: The datetime feature is converted into a datetime object for easier manipulation.
- **Missing Data**: Missing values are imputed using the mean of each column.
- **Feature Scaling**: Features are scaled using StandardScaler to ensure all features contribute equally during model training.

Example code for preprocessing:

```
from sklearn.preprocessing import StandardScaler

import pandas as pd


# Handle missing values and scale features

data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()

scaled_data = scaler.fit_transform(data.drop(columns=['soil_moisture']))
```

2. Exploratory Data Analysis (EDA)

EDA is performed to uncover patterns, trends, and insights in the data.

- **Correlation Heatmap**: Visualized the relationships between different features using a correlation matrix.
- **Feature Distribution**: Plotted histograms to understand the distribution of soil moisture and other variables.
- **Temporal Analysis**: Visualized trends over time to understand how soil moisture and temperature vary.

Example of plotting soil moisture vs temperature:

```
import matplotlib.pyplot as plt


plt.figure(figsize=(10, 6))
plt.plot(data['datetime'], data['soil_moisture'], label='Soil Moisture', color='blue')
plt.plot(data['datetime'], data['soil_temp'], label='Soil Temperature', color='orange')
plt.legend()
plt.title('Temporal Trends of Soil Moisture and Temperature')
plt.xlabel('DateTime')
plt.ylabel('Values')
plt.show()
```

3. Feature Selection

Feature selection is done to choose the most relevant predictors for the model.

- **Random Forest Feature Importance**: Used a Random Forest model to identify the most significant spectral bands influencing soil moisture prediction.
- **Removing Irrelevant Features**: Hyperspectral data features were chosen based on their importance scores from the model.

Example of feature importance extraction:

```
from sklearn.ensemble import RandomForestRegressor


rf = RandomForestRegressor(n_estimators=100)
rf.fit(X_train, y_train)
feature_importances = pd.Series(rf.feature_importances_,
index=X_train.columns).sort_values(ascending=False)
```

4. Model Building and Evaluation

- **Model Selection**: A **Random Forest Regressor** is chosen for its ability to handle complex, nonlinear relationships between the features and target variable (soil moisture).
- **Training the Model**: The data is split into a training and test set using train_test_split. The model is then trained on the training data.
- **Evaluation Metrics**: The model's performance is evaluated using **Mean Absolute Error (MAE)** and **R-squared (R^2)** on the test set.

Example of model evaluation:

```
from sklearn.metrics import mean_absolute_error, r2_score


y_pred = rf.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f'MAE: {mae}, R2: {r2}')
```

5. Hyperparameter Tuning

To improve model performance, **GridSearchCV** or **RandomizedSearchCV** is used to tune the hyperparameters of the model.

- **Grid Search**: A range of hyperparameters (e.g., n_estimators, max_depth) is specified to find the optimal configuration.
- **Random Search**: If grid search is too slow, **RandomizedSearchCV** is used to perform a faster search by randomly selecting a subset of the search space.

Example code for RandomizedSearchCV:

```
from sklearn.model_selection import RandomizedSearchCV

from scipy.stats import randint


param_dist = {

   'n_estimators': randint(50, 200),

   'max_depth': [10, 20, 30, None],

   'min_samples_split': randint(2, 10)

}


random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_dist, n_iter=10, cv=3, n_jobs=-1)

random_search.fit(X_train, y_train)


print(f"Best Parameters: {random_search.best_params_}")
```

6. Final Model Evaluation

After tuning the hyperparameters, the model is retrained using the best parameters. Its performance is again evaluated on the test set.

- **Model Metrics**: The **MAE** and **R²** are recalculated after hyperparameter tuning to evaluate improvements.

Example of final evaluation:

```
y_pred_best = random_search.best_estimator_.predict(X_test)

mae_best = mean_absolute_error(y_test, y_pred_best)

r2_best = r2_score(y_test, y_pred_best)


print(f'After Tuning - MAE: {mae_best}')

print(f'After Tuning - R2: {r2_best}')
```

7. Predictions

The trained model can now be used to predict soil moisture levels for new, unseen data. The predictions are made using the predict method of the trained model.

Example:

```
# Assuming you have new data `X_new`

predictions = random_search.best_estimator_.predict(X_new)
```

8. Conclusion and Next Steps

- **Model Performance**: The Random Forest model can be used to predict soil moisture with reasonable accuracy, as evaluated by MAE and R^2.
- **Model Deployment**: The model can be deployed to make real-time predictions in agriculture, helping farmers optimize irrigation schedules.
- **Future Work**: Further improvements can include experimenting with other models (e.g., XGBoost), adding more features (e.g., soil type, weather data), or applying deep learning models.

References:

- **Random Forest Regressor**: A powerful ensemble method for regression tasks.
- **GridSearchCV & RandomizedSearchCV**: Hyperparameter tuning techniques to optimize model performance.

- **Exploratory Data Analysis (EDA)**: Crucial for uncovering patterns and relationships in the data before modeling.