Crop Recommendation Model – Detailed Documentation

1. Overview

This script trains a multi-output neural network to predict:

1. **Soil type** (classification)
2. **Recommended crops** (multi-label classification)

Inputs include numeric features (N, P, K, rainfall) and categorical features (region, weather, season). The model dynamically handles input sizes based on feature selection and output sizes based on target encoding.

---

2. Reproducibility

```
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
```

- Ensures repeatable results. - `SEED = 42` is fixed. - Seeds for `random`, `numpy`, `tensorflow`.

---

3. Dataset Loading

```
dataset_path = "..."
with open(dataset_path, 'r') as file:
    data = json.load(file)
df = pd.DataFrame(data)
```

- Loads JSON dataset. - Converts to Pandas DataFrame. - Checks for empty DataFrame and prints dimensions.

---

4. Rainfall Parsing

```
def extract_rainfall_value(rainfall_str):
    match = re.match(r'(\d+)-(\d+)', rainfall_str)
```

```
    if match:
        return random.randint(int(match.group(1)), int(match.group(2)))
    return np.nan
```

- Extracts numeric rainfall from string ranges. - Uses random integer in range for approximate numeric value. - Applied to the rainfall column.

---

5. Encoding Categorical Features

```
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
X_cat = encoder.fit_transform(df[['region', 'weather', 'season']])
```

- One-hot encodes categorical features. - `sparse_output=False` ensures dense matrix. - `handle_unknown='ignore'` avoids errors on unseen categories. - Converts to DataFrame with feature names.

---

6. NPK Lookup Tree

```
npk_data = df[['N', 'P', 'K']].values
npk_tree = KDTree(npk_data)
```

- Builds a KDTree for fast nearest neighbor retrieval. - Allows closest NPK match for approximate inputs.

---

7. Target Encoding

```
soil_encoder = LabelEncoder()
y_soil = soil_encoder.fit_transform(df['soil_type'])

crop_mlb = MultiLabelBinarizer()
y_crops = crop_mlb.fit_transform(df['recommended_crops'])
```

- Soil type: label-encoded for classification. - Recommended crops: multi-label binarized. - Supports dynamic number of crops.

---

8. Feature Combination

```
X = pd.concat([X_cat, df[['N', 'P', 'K', 'rainfall']]], axis=1)
```

- Combines categorical (one-hot) and numeric features. - `X.shape[1]` is the dynamic input size for the model.

---

9. Train/Test Split

```
X_train, X_test, y_train_soil, y_test_soil, y_train_crops, y_test_crops =
train_test_split(
    X, y_soil, y_crops, test_size=0.2, random_state=42)
```

- Splits dataset into train/test. - `random_state=42` ensures reproducibility.

---

10. Feature Scaling

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- Scales numeric values to 0–1. - Ensures neural network convergence.

---

11. Model Architecture (Functional API)

```
input_layer = Input(shape=(X_train_scaled.shape[1],))
hidden_layer = Dense(128, activation='relu')(input_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.3)(hidden_layer)
hidden_layer = Dense(64, activation='relu')(hidden_layer)
hidden_layer = BatchNormalization()(hidden_layer)
hidden_layer = Dropout(0.3)(hidden_layer)
hidden_layer = Dense(32, activation='relu')(hidden_layer)

soil_type_output = Dense(len(np.unique(y_soil)), activation='softmax',
name='soil_type_output')(hidden_layer)
recommended_crops_output = Dense(y_crops.shape[1], activation='sigmoid',
name='recommended_crops_output')(hidden_layer)
```

```
model = Model(inputs=input_layer, outputs=[soil_type_output,
recommended_crops_output])
```

Explanation: - **Dense layers:** Fully connected layers extract complex patterns. - **BatchNormalization:** Normalizes outputs for faster training. - **Dropout:** Prevents overfitting. - **Softmax (soil_type):** Produces probabilities summing to 1. - **Sigmoid (crops):** Independent probabilities for multi-label outputs. - Input layer adapts dynamically to `X.shape[1]`. - Output layers adapt to number of soil types / crops.

---

12. Model Compilation

```
model.compile(
    optimizer='adam',
    loss={'soil_type_output': 'sparse_categorical_crossentropy',
'recommended_crops_output': 'binary_crossentropy'},
    loss_weights={'soil_type_output': 1.2, 'recommended_crops_output': 0.8},
    metrics={'soil_type_output': 'accuracy', 'recommended_crops_output':
'accuracy'}
)
```

- **Optimizer:** Adam for adaptive learning. - **Loss functions:** - Soil: categorical. - Crops: multi-label binary. - **Loss weights:** Adjust relative importance. - **Metrics:** Accuracy for monitoring.

---

13. Model Training

```
history = model.fit(
    X_train_scaled,
    {'soil_type_output': y_train_soil, 'recommended_crops_output':
y_train_crops},
    epochs=150,
    batch_size=16,
    validation_split=0.2,
    verbose=1
)
```

- Training with validation split. - `batch_size=16` balances stability and speed. - Epochs increased for better convergence.

---

14. Saving Model and Preprocessors

```
model.save(model_save_path)
joblib.dump(encoder, encoder_save_path)
joblib.dump(scaler, scaler_save_path)
joblib.dump(crop_mlb, crop_mlb_save_path)
joblib.dump(soil_encoder, soil_encoder_save_path)
```

- Saves trained model in `.keras` format. - Saves preprocessors for inference. - Ensures consistency in production.

---

15. Dynamic Inputs & Outputs

- **Input size:** determined by `X.shape[1]`, automatically adapts when features change.
- **Outputs:** determined by number of unique soil types and number of crops in `MultiLabelBinarizer`.
- To add/remove features: modify `X` before scaling.
- To add/remove outputs: modify targets `y_soil` or `y_crops`.

---

16. Concepts Explained

- **Sequential vs Functional API:** Functional API allows multiple outputs and complex connections.
- **Activations:**
- `ReLU`: introduces non-linearity.
- `Softmax`: normalized probabilities for single-label classification.
- `Sigmoid`: independent probabilities for multi-label classification.
- **Dense → BatchNorm → ReLU → Dropout sequence:** stabilizes learning, avoids overfitting, maintains expressiveness.
- **KDTree:** efficiently finds nearest NPK match for approximate input.

✅Summary

- Preprocessing handles numeric and categorical features, normalizes and encodes them.
- Model uses Functional API for dynamic input/output and multi-task learning.
- Layers and activations are chosen for stability, speed, and generalization.
- Outputs provide soil type classification and crop recommendations.
- Changing input/output sizes requires adjusting `X` and target encodings.

End of Documentation