NAME: DURVA KADAM ROLL NO: 23

Assignment 1: Demonstration of Grid Search and Random Search Hyperparameter Tuning

1. Dataset Selection

- ➔ The Wine dataset from scikit-learn was chosen for this experiment.
- ➔ It consists of 178 samples, each described by 13 chemical properties (features).
- ➔ The target variable represents three different classes of wine.

[1] 1s

```python
from sklearn.datasets import load_wine
X, y = load_wine(return_X_y=True)
print("Shape:", X.shape, "Labels:", set(y))
```

```
Shape: (178, 13) Labels: {np.int64(0), np.int64(1), np.int64(2)}
```

[11] 0s

```python
# Step 1: Import libraries
import pandas as pd
from sklearn.datasets import load_wine

# Step 2: Load the dataset
wine = load_wine()
X, y = wine.data, wine.target

# Step 3: Convert to DataFrame
wine_df = pd.DataFrame(X, columns=wine.feature_names)
wine_df['target'] = y

# Step 4: Display dataset shape and unique classes
print("Shape of dataset:", wine_df.shape)
print("Unique target labels:", wine_df['target'].unique())

# Step 5: Display first 10 rows
print("\nFirst 10 rows of the Wine dataset:")
wine_df.head(10)
```

```
Shape of dataset: (178, 14)
Unique target labels: [0 1 2]

First 10 rows of the Wine dataset:
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	ta
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	
5	14.20	1.76	2.45	15.2	112.0	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450.0	
6	14.39	1.87	2.45	14.6	96.0	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290.0	
7	14.06	2.15	2.61	17.6	121.0	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295.0	
8	14.83	1.64	2.17	14.0	97.0	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045.0	
9	13.86	1.35	2.27	16.0	98.0	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045.0	

NAME: DURVA KADAM ROLL NO: 23

2. Model Selection

- Logistic Regression was chosen as the classification algorithm.
- This model is widely used for multiclass problems, is interpretable, and has several hyperparameters that significantly affect performance.
- The hyperparameters considered for tuning were:
    - **C**: Regularization strength (inverse of penalty).
    - **Penalty**: Type of regularization (L1 or L2).
    - **Solver**: Optimization algorithm used to fit the model.
    - **Multi-class strategy**: One-vs-Rest (OvR) or Multinomial.

```
[2]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
[3]
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(max_iter=5000)
```

3. Grid Search Hyperparameter Tuning

- Grid Search was applied to exhaustively evaluate all possible combinations of selected hyperparameters.
- Five-fold cross-validation was used to assess model performance for each combination.
- The best parameter set was identified as: C = 100, penalty = L1, solver = liblinear, multi_class = OvR.
- The corresponding best cross-validation score was 97.2%.

```
[9]
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import GridSearchCV

param_grid = [
    {   # For liblinear, only ovr is valid
        'C': [0.01, 0.1, 1, 10, 100],
        'solver': ['liblinear'],
        'penalty': ['l1', 'l2'],
        'multi_class': ['ovr']
    },
    {   # For saga, both multinomial and ovr work
        'C': [0.01, 0.1, 1, 10, 100],
        'solver': ['saga'],
        'penalty': ['l1', 'l2'],
        'multi_class': ['ovr', 'multinomial']
    }
]

grid = GridSearchCV(log_reg, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters (Grid Search):", grid.best_params_)
print("Best CV Score:", grid.best_score_)
```

```
Best Parameters (Grid Search): {'C': 100, 'multi_class': 'ovr', 'penalty': 'l1', 'solver': 'liblinear'}
Best CV Score: 0.9721674876847292
```

4. Random Search Hyperparameter Tuning

- ➔ Random Search was applied to sample a limited number of hyperparameter combinations from predefined ranges.
- ➔ Twenty random combinations were tested using five-fold cross-validation.
- ➔ The best parameter set was identified as:
  - ◆ C ≈ 0.298, penalty = L2, solver = liblinear, multi_class = OvR.

  - ◆ The corresponding best cross-validation score was 95.8%.

```
[6]
25s
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

param_dist = {
    'C': np.logspace(-3, 2, 100),   # values between 0.001 and 100
    'solver': ['saga', 'liblinear'],
    'penalty': ['l1', 'l2'],
    'multi_class': ['ovr', 'multinomial']
}

rand = RandomizedSearchCV(
    log_reg, param_distributions=param_dist,
    n_iter=20, cv=5, scoring='accuracy', random_state=42
)
rand.fit(X_train, y_train)

print("Best Parameters (Random Search):", rand.best_params_)
print("Best CV Score:", rand.best_score_)

Best Parameters (Random Search): {'solver': 'liblinear', 'penalty': 'l2', 'multi_class': 'ovr', 'C': np.float64(0.298364724028334)}
Best CV Score: 0.9583743842364532
```

5. Results Comparison

- Grid Search achieved slightly higher performance (97.2%) compared to Random Search (95.8%).
- Grid Search was more exhaustive but computationally more expensive.
- Random Search was faster and more efficient, but less thorough.
- Both approaches produced highly accurate models, confirming the effectiveness of Logistic Regression on this dataset.

```
[10]
 0s
from sklearn.metrics import accuracy_score, classification_report

# Grid Search model
y_pred_grid = grid.best_estimator_.predict(X_test)
print("Grid Search Test Accuracy:", accuracy_score(y_test, y_pred_grid))
print(classification_report(y_test, y_pred_grid))

# Random Search model
y_pred_rand = rand.best_estimator_.predict(X_test)
print("Random Search Test Accuracy:", accuracy_score(y_test, y_pred_rand))
print(classification_report(y_test, y_pred_rand))
```

```
Grid Search Test Accuracy: 0.9166666666666666
              precision    recall  f1-score   support

           0       0.86      1.00      0.92        12
           1       0.92      0.86      0.89        14
           2       1.00      0.90      0.95        10

    accuracy                           0.92        36
   macro avg       0.93      0.92      0.92        36
weighted avg       0.92      0.92      0.92        36

Random Search Test Accuracy: 0.9722222222222222
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.93      1.00      0.97        14
           2       1.00      0.90      0.95        10

    accuracy                           0.97        36
   macro avg       0.98      0.97      0.97        36
weighted avg       0.97      0.97      0.97        36
```

What can I help you build?

6. Conclusion

- ➔ Grid Search: Achieved a CV score of 97.2% but a test accuracy of ~91.7%, showing strong training/validation performance but slightly lower generalization.
- ➔ Random Search: Achieved a CV score of 95.8% but test accuracy of ~97.2%, indicating better generalization to unseen data.
- ➔ Conclusion: Exhaustive hyperparameter search (Grid Search) does not always guarantee the best test performance. Random Search can be more efficient and may generalize better in practice.