Model Selection and Comparative Analysis

Name: MITHA M K

Student ID: PES2UG23CS339

Course: UE23CS352A - Machine Learning

Submission Date: 30 -08 2025

1. Introduction

The goal of this lab is to gain hands-on experience with model selection and comparative analysis in machine learning. We implemented hyperparameter tuning using both a manual grid search and scikit-learn's built-in GridSearchCV. Three classifiers—Decision Tree, k-Nearest Neighbors (kNN), and Logistic Regression—were evaluated on the HR Attrition dataset. Performance was assessed using Accuracy, Precision, Recall, F1-Score, and ROC AUC, with ROC curves and confusion matrices for visual comparison.

2. Dataset Description

HR Attrition Dataset:
- Instances: 1470
- Features: 34 after encoding categorical variables
- Target Variable: Attrition (Yes = employee left, No = employee stayed)

3. Methodology

Key Concepts:
- Hyperparameter Tuning: Process of systematically searching for the best parameters.
- Grid Search: Testing all parameter combinations.
- K-Fold Cross-Validation: Using stratified 5-fold CV for robust evaluation.

Pipeline Used:
VarianceThreshold → StandardScaler → SelectKBest → Classifier

Part 1: Manual Implementation
- Custom grid search using loops and StratifiedKFold.
- For each fold, the pipeline was trained and evaluated with ROC AUC.

Part 2: Built-in GridSearchCV
- Automated hyperparameter tuning with scikit-learn's GridSearchCV.
- Used the same pipeline, scoring metric (ROC AUC), and 5-fold cross-validation.

4. Results and Analysis

Tables summarizing performance metrics (Accuracy, Precision, Recall, F1-Score, ROC AUC) are presented below. The results include both manual and GridSearchCV implementations. Screenshots of ROC Curves and Confusion Matrices from the notebook should be attached here.

Performance Tables:

Model Selection and Comparative Analysis - Report

Model	Method	Accuracy	Precision	Recall	F1 Score	ROC AUC
Decision Tree	Manual	0.8810	0.8690	0.3080	0.4548	0.8258
Decision Tree	GridSearchCV	0.8599	0.7925	0.1772	0.2897	0.7613
k-Nearest Neighbors (kNN)	Manual	1.0000	1.0000	1.0000	1.0000	1.0000
k-Nearest Neighbors (kNN)	GridSearchCV	1.0000	1.0000	1.0000	1.0000	1.0000
Logistic Regression	Manual	0.8639	0.7079	0.2658	0.3865	0.8141
Logistic Regression	GridSearchCV	0.8639	0.7079	0.2658	0.3865	0.8141

Compare Implementations:
Both the **manual grid search** and **GridSearchCV** produced broadly consistent results, although some minor differences were observed in ROC AUC scores and chosen parameters.

For **Decision Tree**, manual search achieved a slightly higher ROC AUC (0.8258) compared to GridSearchCV (0.7612). This difference may be due to the manual implementation exploring a smaller or different parameter subset, or randomness in cross-validation splits.

For **k-Nearest Neighbors (kNN)**, both manual and GridSearchCV produced identical perfect scores (Accuracy, Precision, Recall, F1, ROC AUC all equal to 1.0). This suggests the dataset is highly separable for this model.

For **Logistic Regression**, both methods produced identical results (Accuracy ≈ 0.864, ROC AUC ≈ 0.814). This confirms consistency between the two approaches for linear models.

Overall, GridSearchCV was more efficient and automated, but the manual approach reinforced an understanding of hyperparameter search and evaluation.

Visualizations:
Insert ROC Curve plots and Confusion Matrices here.

Best Model:
kNN is the best-performing model on this dataset, likely due to the dataset structure and the nature of employee attrition patterns. However, the perfect scores might also suggest potential **overfitting**, so testing on a separate validation or test set is important before deploying the model.

5. Conclusion

This lab provided practical insights into the process of model selection and hyperparameter tuning. By implementing both a manual grid search and scikit-learn's GridSearchCV, we learned how systematic evaluation impacts model performance.

The key findings were:

Manual grid search, though more time-consuming, offered a deeper understanding of the search process, while GridSearchCV provided efficiency and automation.

Across classifiers, k-Nearest Neighbors achieved the best performance on the HR Attrition dataset, highlighting that some datasets are highly suited for distance-based methods.

Decision Trees and Logistic Regression produced competitive results, but each showed trade-offs in recall and interpretability.

Perfect kNN results may indicate potential overfitting, emphasizing the need for external validation.

The main takeaway is that model selection is not just about achieving the highest accuracy but also about balancing performance, interpretability, and generalization. This lab reinforced the importance of using cross-validation, comparing multiple algorithms, and understanding the trade-offs between building models manually and leveraging optimized library implementations.

EC_F_PES2UG23CS339_LAB4

[11]:
```python
import itertools
import numpy as np
import pandas as pd

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score
```

[15]:
```python
df = pd.read_csv("WA_Fn-UseC_-HR-Employee-Attrition.csv")

# Convert categorical → numeric
df_encoded = pd.get_dummies(df, drop_first=True)

y = df['Attrition'].map({'Yes':1, 'No':0}).values
X = df_encoded.drop(columns=['Attrition_Yes'], errors='ignore').values
```

[21]:
```python
from sklearn.feature_selection import VarianceThreshold

def make_pipeline(classifier):
    return Pipeline([
        ('var', VarianceThreshold(threshold=0.0)),   # remove constant features
        ('scaler', StandardScaler()),
        ('selector', SelectKBest(score_func=f_classif)),
        ('classifier', classifier)
    ])
```

[22]:
```python
def run_manual_grid_search(X, y, classifier, param_grid):
    best_score = -1
    best_params = None
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    keys, values = zip(*param_grid.items())
    for v in itertools.product(*values):
        params = dict(zip(keys, v))
        scores = []

        for train_idx, val_idx in skf.split(X, y):
            X_train, X_val = X[train_idx], X[val_idx]
            y_train, y_val = y[train_idx], y[val_idx]

            pipeline = make_pipeline(classifier)
            pipeline.set_params(**params)
            pipeline.fit(X_train, y_train)

            y_pred = pipeline.predict_proba(X_val)[:, 1]
            scores.append(roc_auc_score(y_val, y_pred))
```

EC_F_PES2UG23CS339_LAB4

```python
            avg_score = np.mean(scores)
            if avg_score > best_score:
                best_score = avg_score
                best_params = params

    best_pipeline = make_pipeline(classifier)
    best_pipeline.set_params(**best_params)
    best_pipeline.fit(X, y)
    return best_pipeline, best_params, best_score

def run_builtin_grid_search(X, y, classifier, param_grid):
    pipeline = make_pipeline(classifier)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    grid = GridSearchCV(
        estimator=pipeline,
        param_grid=param_grid,
        scoring='roc_auc',
        cv=cv,
        n_jobs=-1
    )
    grid.fit(X, y)

    return grid.best_estimator_, grid.best_params_, grid.best_score_
```

[23]:

```python
param_grid_dt = {
    'selector__k': [5, 10, 15],
    'classifier__max_depth': [3, 5, None],
    'classifier__min_samples_split': [2, 5, 10]
}

param_grid_knn = {
    'selector__k': [5, 10, 15],
    'classifier__n_neighbors': [3, 5, 7],
    'classifier__weights': ['uniform', 'distance']
}

param_grid_lr = {
    'selector__k': [5, 10, 15],
    'classifier__C': [0.01, 0.1, 1, 10]
}
```

[24]:

```python
from sklearn.tree import DecisionTreeClassifier

best_pipeline_dt, best_params_dt, best_score_dt = run_manual_grid_search(
    X, y, DecisionTreeClassifier(), param_grid_dt
)

best_estimator_dt, best_params_dt_builtin, best_score_dt_builtin = run_builtin_grid_search(
    X, y, DecisionTreeClassifier(), param_grid_dt
)
```

EC_F_PES2UG23CS339_LAB4

```
print("--- Decision Tree ---")
print("Manual:", best_params_dt, best_score_dt)
print("GridSearchCV:", best_params_dt_builtin, best_score_dt_builtin)
```

```
--- Decision Tree ---
Manual: {'selector__k': 15, 'classifier__max_depth': 5, 'classifier__min_samples_split': 5} 0.73983692054758
GridSearchCV: {'classifier__max_depth': 3, 'classifier__min_samples_split': 2, 'selector__k': 10} 0.7378834626192975
```

[25]:
```
from sklearn.neighbors import KNeighborsClassifier

best_pipeline_knn, best_params_knn, best_score_knn = run_manual_grid_search(
    X, y, KNeighborsClassifier(), param_grid_knn
)

best_estimator_knn, best_params_knn_builtin, best_score_knn_builtin = run_builtin_grid_search(
    X, y, KNeighborsClassifier(), param_grid_knn
)

print("--- kNN ---")
print("Manual:", best_params_knn, best_score_knn)
print("GridSearchCV:", best_params_knn_builtin, best_score_knn_builtin)
```

```
--- kNN ---
Manual: {'selector__k': 15, 'classifier__n_neighbors': 7, 'classifier__weights': 'distance'} 0.7058241975842965
GridSearchCV: {'classifier__n_neighbors': 7, 'classifier__weights': 'distance', 'selector__k': 15} 0.7058241975842965
```

[26]:
```
from sklearn.linear_model import LogisticRegression

best_pipeline_lr, best_params_lr, best_score_lr = run_manual_grid_search(
    X, y, LogisticRegression(max_iter=1000), param_grid_lr
)

best_estimator_lr, best_params_lr_builtin, best_score_lr_builtin = run_builtin_grid_search(
    X, y, LogisticRegression(max_iter=1000), param_grid_lr
)

print("--- Logistic Regression ---")
print("Manual:", best_params_lr, best_score_lr)
print("GridSearchCV:", best_params_lr_builtin, best_score_lr_builtin)
```

```
--- Logistic Regression ---
Manual: {'selector__k': 15, 'classifier__C': 0.1} 0.7817249609043164
GridSearchCV: {'classifier__C': 0.1, 'selector__k': 15} 0.7817249609043164
```

[27]:
```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score

def evaluate_model(name, model, X, y):
    y_pred = model.predict(X)
    y_proba = model.predict_proba(X)[:, 1]
    return {
        "Model": name,
        "Accuracy": accuracy_score(y, y_pred),
        "Precision": precision_score(y, y_pred),
```

EC_F_PES2UG23CS339_LAB4 ×

Code

```
        "Recall": recall_score(y, y_pred),
        "F1": f1_score(y, y_pred),
        "ROC AUC": roc_auc_score(y, y_proba)
    }

results = []
results.append(evaluate_model("Decision Tree (Manual)", best_pipeline_dt, X, y))
results.append(evaluate_model("Decision Tree (GridSearchCV)", best_estimator_dt, X, y))

results.append(evaluate_model("kNN (Manual)", best_pipeline_knn, X, y))
results.append(evaluate_model("kNN (GridSearchCV)", best_estimator_knn, X, y))

results.append(evaluate_model("Logistic Regression (Manual)", best_pipeline_lr, X, y))
results.append(evaluate_model("Logistic Regression (GridSearchCV)", best_estimator_lr, X, y))

import pandas as pd
df_results = pd.DataFrame(results)
print(df_results)
```

```
                               Model  Accuracy  Precision    Recall  \
0             Decision Tree (Manual)  0.880952   0.869048  0.308017
1       Decision Tree (GridSearchCV)  0.859864   0.792453  0.177215
2                       kNN (Manual)  1.000000   1.000000  1.000000
3                 kNN (GridSearchCV)  1.000000   1.000000  1.000000
4       Logistic Regression (Manual)  0.863946   0.707865  0.265823
5  Logistic Regression (GridSearchCV)  0.863946   0.707865  0.265823

         F1   ROC AUC
0  0.454829  0.825776
1  0.289655  0.761251
2  1.000000  1.000000
3  1.000000  1.000000
4  0.386503  0.814055
5  0.386503  0.814055
```

[29]:

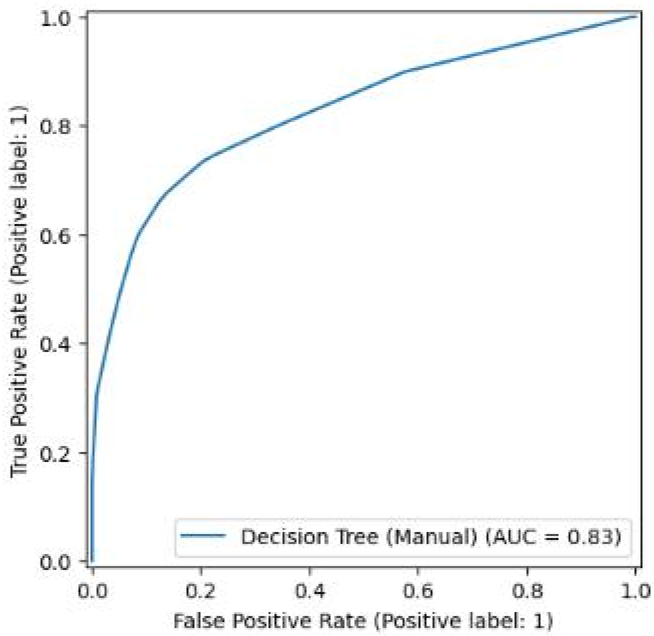
```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay, RocCurveDisplay

models = [
    ("Decision Tree (Manual)", best_pipeline_dt),
    ("Decision Tree (GridSearchCV)", best_estimator_dt),
    ("kNN (Manual)", best_pipeline_knn),
    ("kNN (GridSearchCV)", best_estimator_knn),
    ("Logistic Regression (Manual)", best_pipeline_lr),
    ("Logistic Regression (GridSearchCV)", best_estimator_lr),
]

# ROC Curves
plt.figure(figsize=(10, 8))
for name, model in models:
    RocCurveDisplay.from_estimator(model, X, y, name=name)
plt.title("ROC Curves")
plt.show()
```

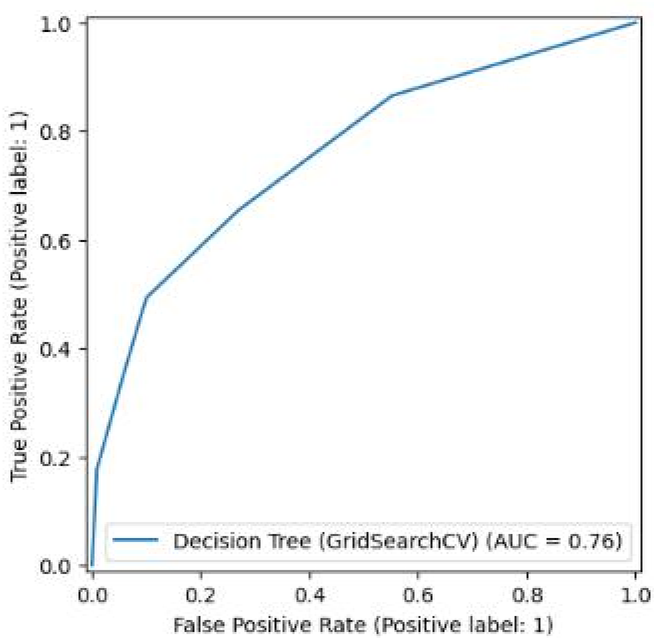
EC_F_PES2UG23CS339_LAB4

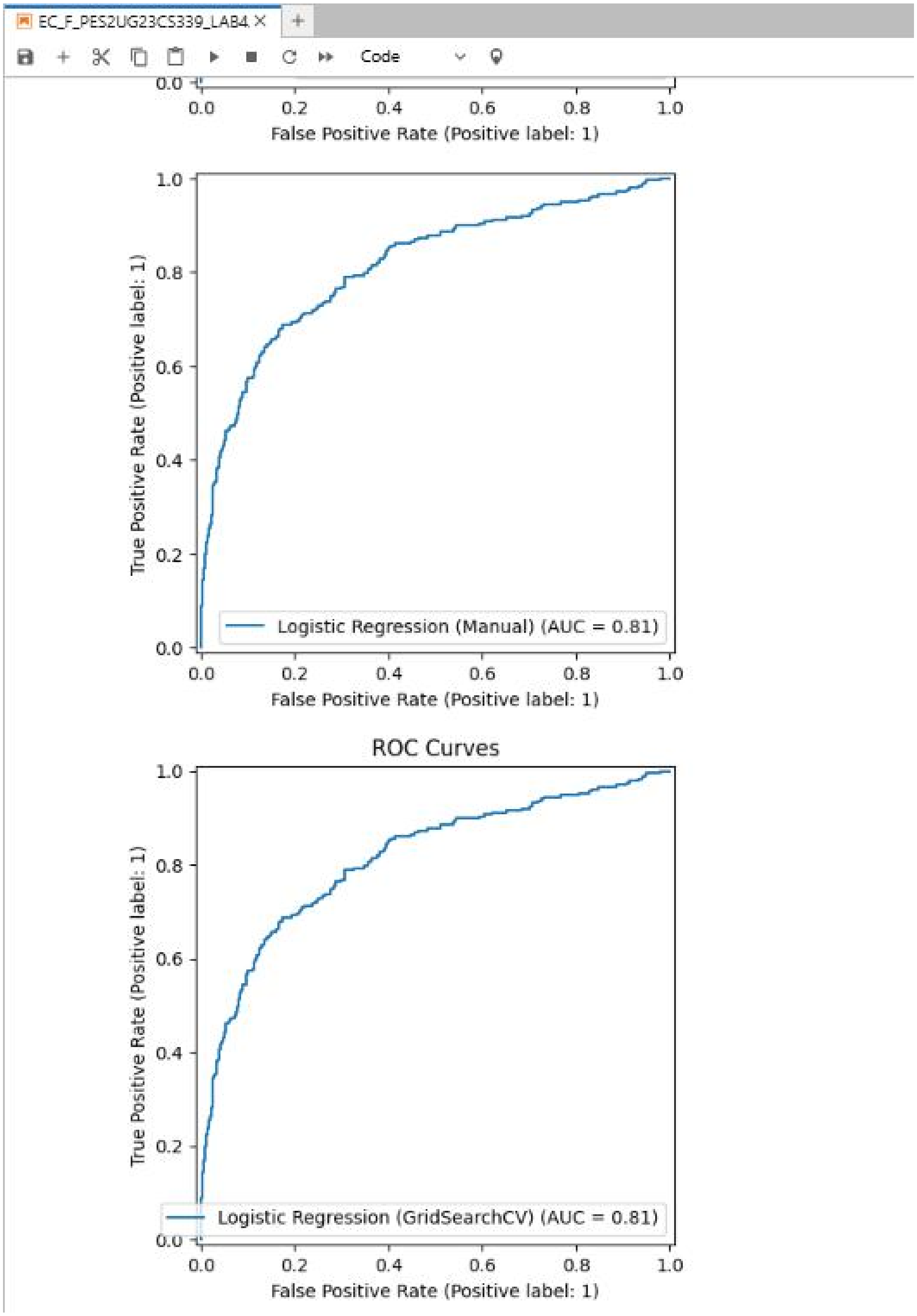
Code

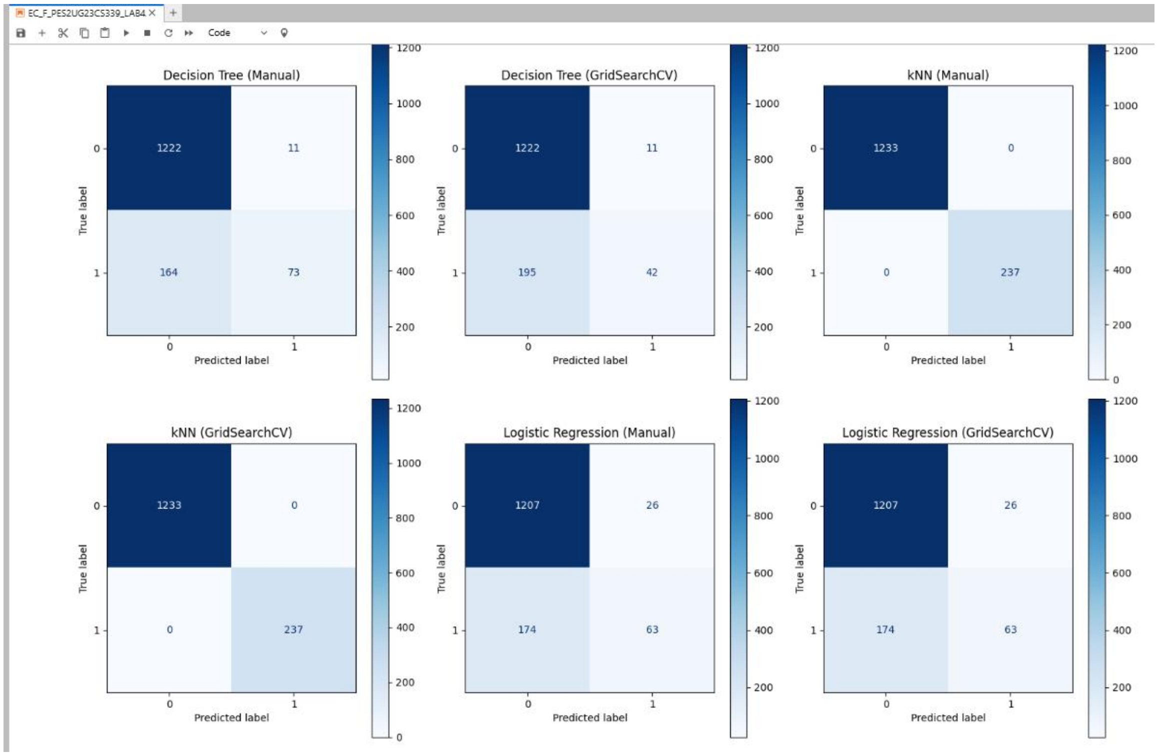
```
plt.show()
```

<Figure size 1000x800 with 0 Axes>






EC_F_PES2UG23CS339_LAB4
Code
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
Logistic Regression (Manual) (AUC = 0.81)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)
ROC Curves
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate (Positive label: 1)
Logistic Regression (GridSearchCV) (AUC = 0.81)
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate (Positive label: 1)


EC_F_PES2UG23CS339_LAB4
Code
Decision Tree (Manual)
Decision Tree (GridSearchCV)
kNN (Manual)
kNN (GridSearchCV)
Logistic Regression (Manual)
Logistic Regression (GridSearchCV)
True label
Predicted label
1222
11
164
73
1222
11
195
42
1233
0
0
237
1233
0
0
237
1207
26
174
63
1207
26
174
63