Theoretical

In [ ]:
```
#What is a Decision Tree, and how does it work
'''
A Decision Tree is a supervised machine learning algorithm used for both classifica
recursively splitting the dataset into subsets based on feature values, forming a t
to a final outcome. The process begins at the root node, which represents the entir
split the data using metrics like Gini Impurity or Information Gain. This splitting
condition, such as a maximum depth or when all samples in a node belong to a single
decision nodes, where further splits occur, and leaf nodes that represent the final
navigates from the root to a leaf node based on feature values, providing a straigh
'''
#What are impurity measures in Decision Trees
'''
Impurity measures in Decision Trees are metrics used to assess how mixed the classe
best feature for splitting the data. The goal is to reduce impurity with each split
most samples belong to a single class. Common impurity measures include Gini Impuri
misclassifying a randomly chosen element based on the class distribution within a n
disorder or uncertainty in a node and is used to maximize Information Gain during s
measure representing the fraction of misclassified samples, though it is less sensi
these impurity measures, Decision Trees can more effectively classify data.
'''
#What is the mathematical formula for Gini Impurity
'''      c
Gini = 1-∑(pi2)
        i=1
 C is Total number of classes.
 Pi is  Proportion of samples belonging to class i in a node.
 '''
#What is the mathematical formula for Entropy
'''        c
Entopy = - ∑ pi log2 (pi)
          i=1

C is Total number of classes.
Pi is  Proportion of samples belonging to class i in a node
log2 is Logarithm base 2.
'''
#What is Information Gain, and how is it used in Decision Trees
'''
Information Gain is a metric used in Decision Trees to measure the effectiveness of
impurity in a dataset. It is based on the concept of Entropy and helps identify the
Higher Information Gain indicates a more informative feature that leads to purer su
Feature Selection: In a decision tree, the feature with the highest Information Gai
criterion at each node.
Tree Growth: The decision tree recursively splits data based on features that maxim
criterion (e.g., maximum depth, minimum samples per leaf) is met.
Reducing Uncertainty: Higher Information Gain indicates that a feature effectively
child nodes with homogenous class labels.
'''


#What is the difference between Gini Impurity and Entropy
'''
Gini Impurity and Entropy are both measures of impurity used in decision tree algor
```

```
splitting the data. Gini Impurity calculates the probability of incorrectly classif
by 1 - ∑ pi2 where pi is the probability of a class. It ranges from 0 (pure node) t
classification) and is computationally faster since it avoids logarithmic calculati
the uncertainty in a system using the formula -∑ pi log2pi ranging from 0 (pure nod
classification). While Entropy provides a more theoretical measure of disorder, it
logarithm function. Gini is preferred in the CART algorithm, whereas Entropy is use
practice, both perform similarly, and the choice between them often depends on comp
default in many implementations like sklearn.DecisionTreeClaasifier.
'''
#What is the mathematical explanation behind Decision Trees
'''
Decision Trees are a fundamental concept in Machine Learning, and their mathematica


Decision Tree Components:
1.Root Node: The topmost node representing the input data.
2.Decision Nodes: Intermediate nodes that split the data based on a feature or attr
3.Leaf Nodes: Terminal nodes representing the predicted class labels or target valu

Mathematical Formulation:
1.Entropy: A measure of uncertainty or randomness in the data, calculated using Sha

H(D) = - ∑ (p(x) * log2(p(x)))

where H(D) is the entropy of the dataset D, p(x) is the probability of each class l


2.Information Gain: The reduction in entropy after splitting the data at a decision
formula:

IG(D, f) = H(D) - ∑ (|Dj|/|D| * H(Dj))

where IG(D, f) is the information gain, H(D) is the entropy of the dataset D, |Dj|
subset, |D| is the total number of instances, and f is the feature used for splitti

3.Gain Ratio: A variant of information gain that takes into account the number of p
using the gain ratio formula:

GR(D, f) = IG(D, f) / ∑ (|Dj|/|D| * log2(|Dj|/|D|))

where GR(D, f) is the gain ratio.

Decision Tree Induction:
1.Recursive Partitioning: The decision tree is constructed recursively by selecting
gain (or gain ratio) at each decision node.
2.Stopping Criteria: The recursion stops when a leaf node is reached, typically whe
class or when a maximum depth is reached.

Mathematical Optimization:
Decision Tree induction can be formulated as an optimization problem, where the goa
information gain (or gain ratio) while minimizing the tree's complexity.

Common Decision Tree Algorithms:
1.ID3 (Iterative Dichotomizer 3): Uses information gain to select features.
2.C4.5: An extension of ID3 that uses gain ratio to select features.
3.CART (Classification and Regression Trees): Uses Gini impurity to select features
```

```
These algorithms and mathematical concepts form the foundation of Decision Trees, a
regression tasks in Machine Learning.
'''
#What is Pre-Pruning in Decision Trees
'''
Pre-Pruning (also called early stopping) is a technique used to prevent a decision
overfitting and improving generalization. Instead of allowing the tree to grow full
stops the tree from expanding beyond a certain point based on predefined conditions
How Pre-Pruning Works
During tree construction, the algorithm evaluates whether to continue splitting a n
a condition is met, the split is prevented, and the node is converted into a leaf n
Maximum Depth (max_depth) – Limits how deep the tree can grow.
Minimum Samples per Split (min_samples_split) – Requires a minimum number of sample
Minimum Samples per Leaf (min_samples_leaf) – Ensures that each leaf has a minimum
Maximum Number of Leaves (max_leaf_nodes) – Restricts the total number of leaf node
Information Gain or Gini Threshold – Stops splitting if the gain is below a certain

Advantages of Pre-Pruning
Prevents Overfitting: By stopping early, the model avoids unnecessary complexity.
Faster Training: Smaller trees require fewer computations.
Better Generalization: The tree is less likely to memorize noise in the training da

Disadvantages of Pre-Pruning
Underfitting Risk: Stopping too early may prevent the tree from capturing important
Choosing the Right Threshold is Hard: If the pre-pruning criteria are too strict, t
'''
#What is Post-Pruning in Decision Trees
'''
Post-Pruning in Decision Trees
Post-Pruning (also called pruning after training) is a technique used to reduce the
removing branches that have little importance. This helps in reducing overfitting a

How Post-Pruning Works
Train the Full Tree: First, the decision tree is allowed to grow completely, creati
the training data.
Evaluate Node Importance: The tree is then analyzed to determine if removing certai
data.
Remove Unnecessary Branches: Branches that do not significantly contribute to class
them with leaf nodes.

There are two main types of post-pruning:

Cost Complexity Pruning (CCP): Removes nodes based on a cost-complexity parameter (
Reduced-Error Pruning: Iteratively removes nodes and checks validation accuracy, ke

Advantages of Post-Pruning
 Better Generalization: Reduces overfitting by removing noisy branches.
 More Interpretable Models: Pruned trees are smaller and easier to understand.
Prevents Unnecessary Complexity: Keeps only meaningful splits that improve accuracy

Disadvantages of Post-Pruning
Computationally Expensive: Requires additional steps after training.
Dependent on Validation Data: Pruning decisions rely on performance on a separate d
'''
#What is the difference between Pre-Pruning and Post-Pruning
```

```
'''
Pre-Pruning and Post-Pruning are techniques used in decision trees to prevent overf
Pre-Pruning (also called early stopping) stops the tree from growing beyond a certa
conditions like maximum depth, minimum samples per split, or minimum information ga
prevents excessive tree growth but risks underfitting if stopped too early. In cont
the tree fully and then removing unnecessary branches based on their impact on accu
Pruning (CCP) or Reduced-Error Pruning. While Post-Pruning is more effective at bal
more expensive as it requires additional validation. Pre-Pruning is faster and easi
patterns, whereas Post-Pruning is more precise and helps retain meaningful splits.
techniques is often used for optimal results.
'''
# What is a Decision Tree Regressor
'''
A Decision Tree Regressor is a supervised machine learning algorithm that predicts
categorical labels. It works by recursively splitting the dataset into smaller regi
Decision Tree Classifier, but instead of classifying data points, it predicts a num
in each leaf node.
How It Works:
Splitting the Data: The algorithm selects a feature and a split point that minimize
or mean absolute error (MAE)).
Recursive Partitioning: The process continues, creating branches until a stopping c
leaf, maximum depth).
Prediction: When a new data point reaches a leaf node, the output is the average of
'''
#What are the advantages and disadvantages of Decision Trees
'''
 Advantages:
Easy to Interpret and Understand – Decision trees are simple and intuitive, making
technical stakeholders.
Handles Both Numerical and Categorical Data – Unlike some algorithms that require d
with mixed data types.
No Need for Feature Scaling – Unlike algorithms like SVM or KNN, decision trees do
Captures Non-Linear Relationships – Decision trees can model complex decision bound
between features and target variables.
Feature Selection is Automatic – The algorithm selects the most important features
selection.
Can Handle Missing Values – Decision trees can handle missing data by making splits
Fast Training and Prediction – Compared to deep learning models, decision trees tra
Can Be Used for Both Classification and Regression – Decision trees are versatile a

Disadvantages:
Prone to Overfitting – Decision trees tend to create deep structures that fit the t
generalization.
High Variance – Small changes in the data can lead to significantly different trees
Greedy Algorithm May Lead to Suboptimal Splits – Decision trees make locally optima
result in the best overall tree.
Biased Towards Dominant Classes – If classes are imbalanced, the tree may be biased
Computationally Expensive for Large Datasets – While small trees train quickly, ver
and memory-intensive.
Cannot Extrapolate for Regression Tasks – Unlike linear regression, decision trees
data.
To mitigate these disadvantages, techniques like pruning (pre-pruning & post-prunin
Gradient Boosting), and hyperparameter tuning can be used. Would you like a compari
'''
#How does a Decision Tree handle missing values
'''
```

```
How Decision Trees Handle Missing Values
Decision Trees can handle missing values in multiple ways, making them robust for d
handling strategies depend on whether the missing values are in features (independe
variable).

 Handling Missing Values in Features
When feature values are missing, Decision Trees can handle them in the following wa

A. Assign the Most Frequent or Mean Value (Imputation)
For categorical variables, replace missing values with the most frequent category.
For numerical variables, replace missing values with the mean or median of the feat
B. Use Surrogate Splits
Instead of discarding data with missing values, decision trees create alternative s
other features that strongly correlate with the missing feature.
Some implementations (like CART) use surrogate variables to decide where a missing
C. Assign to the Most Common Branch
The model can assign missing values to the most frequent branch at the split node.
This is a simple but less accurate approach.
D. Distribute Missing Values Proportionally
Instead of making a single assignment, missing values are distributed across possib
non-missing values.
'''
# How does a Decision Tree handle categorical features
'''
Decision Trees can handle categorical features efficiently by using different strat
algorithms like Linear Regression or SVM, which require numerical inputs, Decision
variables without needing one-hot encoding (in some implementations).
Splitting Criteria for Categorical Features
When a categorical feature is chosen for splitting, Decision Trees determine the be
gain (Entropy), Gini Impurity, or variance reduction (for regression).

A. For Features with Two Categories (Binary Features)
The tree splits the data into two branches based on the two possible values.
Example: If "Gender" has values {Male, Female}, the tree can split into two branche
B. For Features with Multiple Categories (Multiclass Features)
There are two main ways to handle multi-class categorical features:

One-Versus-All Splitting (Best Single Split)

The algorithm chooses the best single category to split at each step.
Example: If "Color" has {Red, Blue, Green, Yellow}, the tree may split as {Red vs.
Multiway Splitting (Separate Branch for Each Category)

The tree creates multiple branches, one for each category.
Example: If "City" has {New York, London, Paris, Tokyo}, each gets a branch.
This works well for small numbers of categories but can make trees very deep and co
'''
# What are some real-world applications of Decision Trees?
'''
Decision Trees are widely used in various industries due to their simplicity, inter
numerical and categorical data. Here are some common real-world applications.
1.Finance & Banking - Credit Risk Assessment,Fraud Detection,Stock Market Predictio
2.Healthcare & Medicine - Disease Diagnosis,Treatment Recommendation,Patient Readmi
3.Retail & E-Commerce - Customer Segmentation,Product Recommendation,Churn Predicti
4.Education & Human Resources - Student Performance Prediction,Hiring & Resume Scre
```

Practical

In [1]:
```
#Write a Python program to train a Decision Tree Classifier on the Iris dataset and
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

In [2]:
```
#Write a Python program to train a Decision Tree Classifier using Gini Impurity as
#importances

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(criterion="gini", random_state=42)
clf.fit(X_train, y_train)

print("Feature Importances:")
for feature, importance in zip(iris.feature_names, clf.feature_importances_):
    print(f"{feature}: {importance:.4f}")
```

```
Feature Importances:
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

In [3]:
```
#Write a Python program to train a Decision Tree Classifier using Entropy as the sp


iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

In [6]:
```
#Write a Python program to train a Decision Tree Regressor on a dataset and evaluat
from sklearn.datasets import load_diabetes
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
data = load_diabetes()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

```
Mean Squared Error: 4976.7978
```

In [8]:
```
!pip install graphviz
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting graphviz
  Downloading graphviz-0.20.3-py3-none-any.whl.metadata (12 kB)
Downloading graphviz-0.20.3-py3-none-any.whl (47 kB)
Installing collected packages: graphviz
Successfully installed graphviz-0.20.3
```

In [9]:
```
# Write a Python program to train a Decision Tree Classifier and visualize the tree
import graphviz
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


iris = datasets.load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


clf = DecisionTreeClassifier(criterion="gini", max_depth=3, random_state=42)
clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
dot_data = export_graphviz(clf, out_file=None,
                           feature_names=iris.feature_names,
                           class_names=iris.target_names,
                           filled=True, rounded=True, special_characters=True)


graph = graphviz.Source(dot_data)
graph.render("decision_tree")
graph.view()
```

```
Model Accuracy: 1.00
```

Out[9]: 'decision_tree.pdf'

```
Error: no "view" mailcap rules found for type "application/pdf"
```

In [10]:

```
# Write a Python program to train a Decision Tree Classifier with a maximum depth o
#grown tree

iris = load_iris()
X, y = iris.data, iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta


clf_limited = DecisionTreeClassifier(max_depth=3, random_state=42)
clf_limited.fit(X_train, y_train)


clf_full = DecisionTreeClassifier(random_state=42)
clf_full.fit(X_train, y_train)


y_pred_limited = clf_limited.predict(X_test)
y_pred_full = clf_full.predict(X_test)


accuracy_limited = accuracy_score(y_test, y_pred_limited)
accuracy_full = accuracy_score(y_test, y_pred_full)

print(f"Accuracy with max depth 3: {accuracy_limited:.4f}")
print(f"Accuracy with fully grown tree: {accuracy_full:.4f}")
```

```
Accuracy with max depth 3: 1.0000
Accuracy with fully grown tree: 1.0000
```

In [13]:

```
#Write a Python program to train a Decision Tree Classifier using min_samples_split


from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
```

```
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf_limited = DecisionTreeClassifier(min_samples_split=5, random_state=42)
clf_limited.fit(X_train, y_train)

clf_default = DecisionTreeClassifier(random_state=42)
clf_default.fit(X_train, y_train)

y_pred_limited = clf_limited.predict(X_test)
y_pred_default = clf_default.predict(X_test)

accuracy_limited = accuracy_score(y_test, y_pred_limited)
accuracy_default = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with min_samples_split=5: {accuracy_limited:.4f}")
print(f"Accuracy with default tree: {accuracy_default:.4f}")
```

```
Accuracy with min_samples_split=5: 1.0000
Accuracy with default tree: 1.0000
```

In [14]:

```
#Write a Python program to apply feature scaling before training a Decision Tree Cl
#unscaled data

from sklearn.preprocessing import StandardScaler

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Without Feature Scaling
clf_unscaled = DecisionTreeClassifier(random_state=42)
clf_unscaled.fit(X_train, y_train)
y_pred_unscaled = clf_unscaled.predict(X_test)
accuracy_unscaled = accuracy_score(y_test, y_pred_unscaled)

# With Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf_scaled = DecisionTreeClassifier(random_state=42)
clf_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = clf_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)

print(f"Accuracy without scaling: {accuracy_unscaled:.4f}")
print(f"Accuracy with scaling: {accuracy_scaled:.4f}")
```

```
Accuracy without scaling: 1.0000
Accuracy with scaling: 1.0000
```

In [15]:

```
#Write a Python program to train a Decision Tree Classifier using One-vs-Rest (OvR)

from sklearn.multiclass import OneVsRestClassifier
```

```
iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf_ovr = OneVsRestClassifier(DecisionTreeClassifier(random_state=42))
clf_ovr.fit(X_train, y_train)

y_pred = clf_ovr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy using One-vs-Rest strategy: {accuracy:.4f}")
```

```
Accuracy using One-vs-Rest strategy: 1.0000
```

In [16]:
```
# Write a Python program to train a Decision Tree Classifier and display the featur


iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

feature_importances = clf.feature_importances_

for feature, importance in zip(iris.feature_names, feature_importances):
    print(f"{feature}: {importance:.4f}")
```

```
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

In [17]:
```
#Write a Python program to train a Decision Tree Regressor with max_depth=5 and com
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

regressor_limited = DecisionTreeRegressor(max_depth=5, random_state=42)
regressor_limited.fit(X_train, y_train)

regressor_unrestricted = DecisionTreeRegressor(random_state=42)
regressor_unrestricted.fit(X_train, y_train)
```

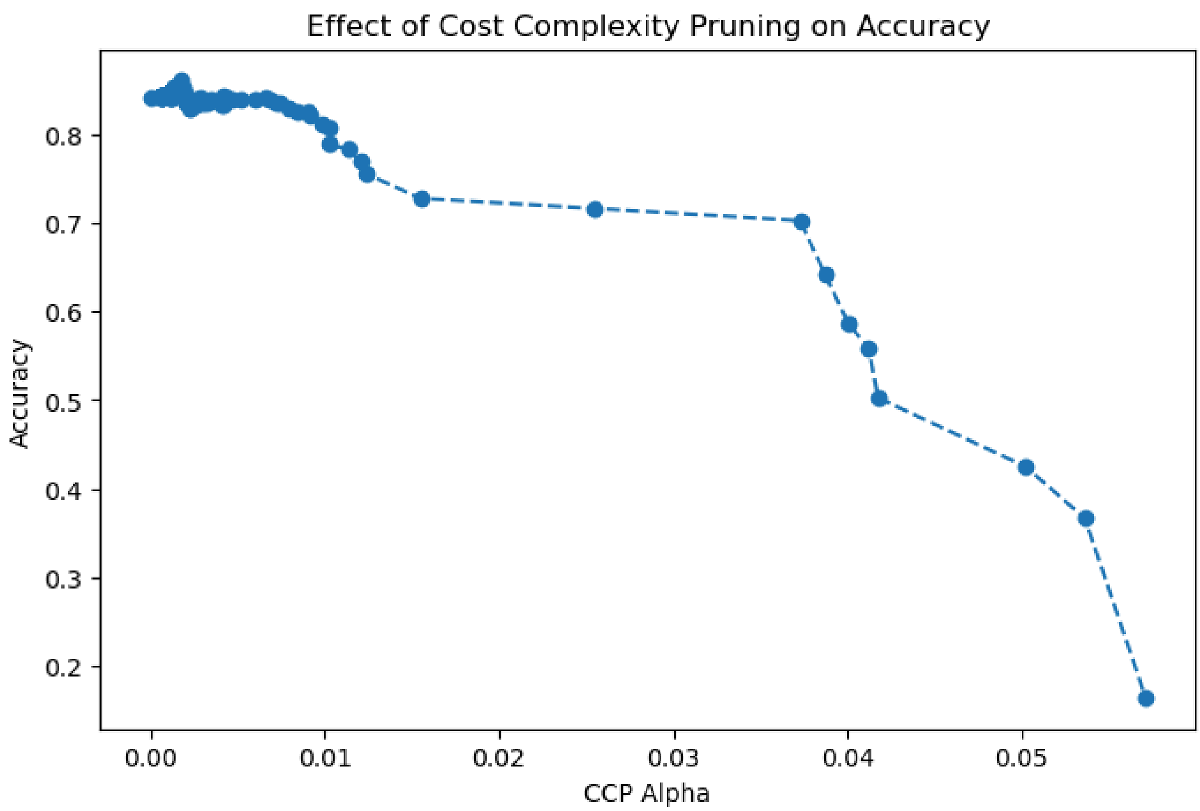
```
y_pred_limited = regressor_limited.predict(X_test)
y_pred_unrestricted = regressor_unrestricted.predict(X_test)

mse_limited = mean_squared_error(y_test, y_pred_limited)
mse_unrestricted = mean_squared_error(y_test, y_pred_unrestricted)

print(f"MSE (max_depth=5): {mse_limited:.2f}")
print(f"MSE (unrestricted tree): {mse_unrestricted:.2f}")

plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred_limited, label="max_depth=5", alpha=0.6)
plt.scatter(y_test, y_pred_unrestricted, label="Unrestricted", alpha=0.6)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Decision Tree Regression Performance")
plt.legend()
plt.show()
```

```
MSE (max_depth=5): 3526.02
MSE (unrestricted tree): 4976.80
```

In [18]:

```
#Write a Python program to train a Decision Tree Classifier, apply Cost Complexity
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(random_state=42)
```

```
clf.fit(X_train, y_train)

path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas[:-1]

accuracies = []
for alpha in ccp_alphas:
    pruned_clf = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    pruned_clf.fit(X_train, y_train)
    y_pred = pruned_clf.predict(X_test)
    accuracies.append(accuracy_score(y_test, y_pred))

plt.figure(figsize=(8, 5))
plt.plot(ccp_alphas, accuracies, marker='o', linestyle='dashed')
plt.xlabel("CCP Alpha")
plt.ylabel("Accuracy")
plt.title("Effect of Cost Complexity Pruning on Accuracy")
plt.show()
```

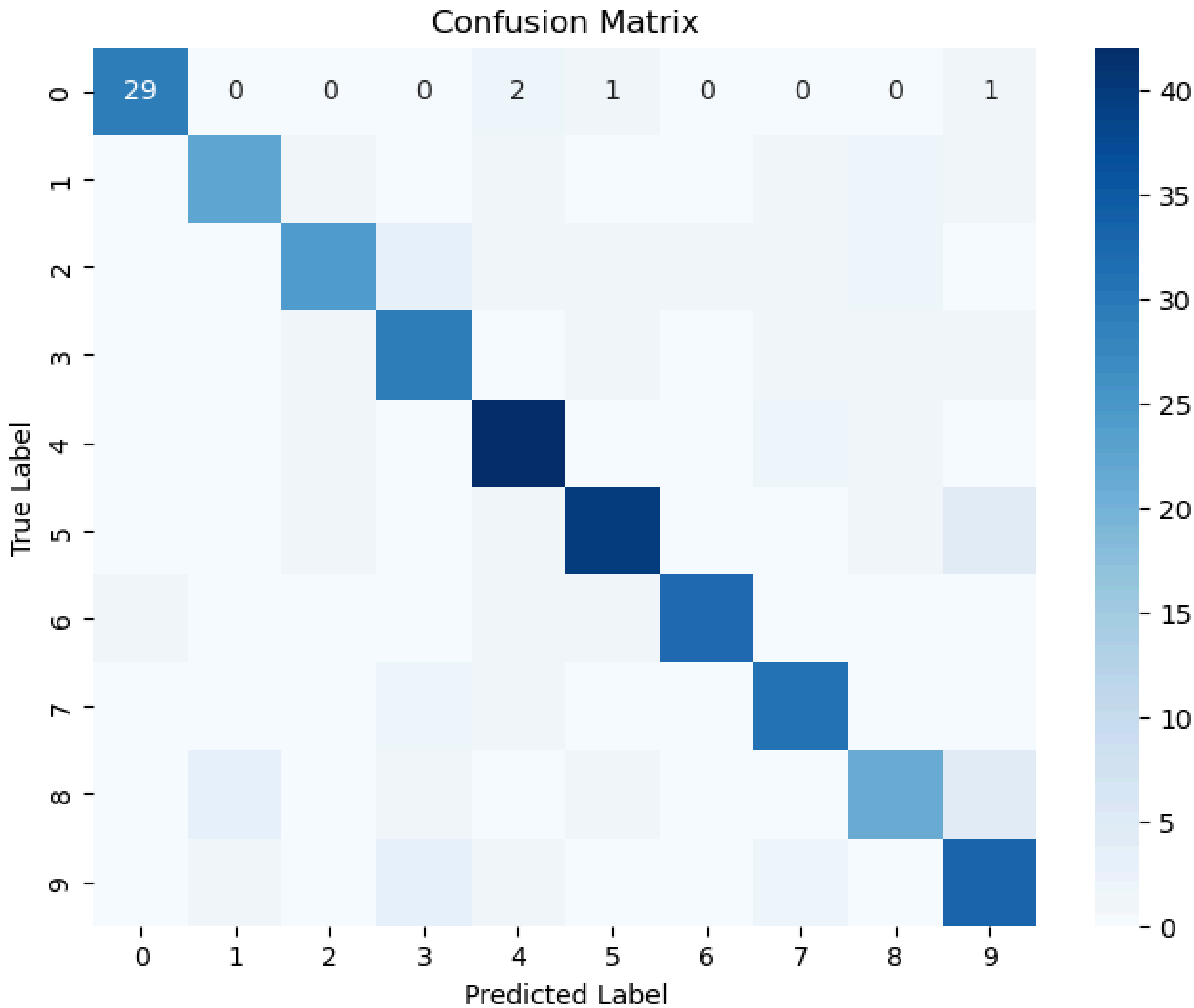


In [19]:
```
#Write a Python program to train a Decision Tree Classifier and evaluate its perfor
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_score, recall_score, f1_score

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(random_state=42)
```

```
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Precision: 0.8447496596970281
Recall: 0.8359873796461651
F1-Score: 0.8384595552746351
```

In [20]:
```
#Write a Python program to train a Decision Tree Classifier and visualize the confu
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=digits.target_names,
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



In [21]:
```
#Write a Python program to train a Decision Tree Classifier and use GridSearchCV to
#for max_depth and min_samples_split
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier

digits = load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

param_grid = {
    'max_depth': [3, 5, 10, None],
    'min_samples_split': [2, 5, 10]
}

clf = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_clf = grid_search.best_estimator_
accuracy = best_clf.score(X_test, y_test)
```

```
print("Best Parameters:", grid_search.best_params_)
print("Best Model Accuracy:", accuracy)
```

```
Best Parameters: {'max_depth': None, 'min_samples_split': 2}
Best Model Accuracy: 0.8416666666666667
```

In [ ]: