ML-PS4

Duoshu Xu

Partnered with Jae Hu

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score, KFold
from sklearn.cross_decomposition import PLSRegression
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix
```

Question 1

```
# load the dataset
file_path = "/Users/kevinxu/Documents/GitHub/ML-PS4/Data-College.csv"
college_df = pd.read_csv(file_path)
```

Question 1a

```
# drop the first column which is the name of the university
college_df = college_df.drop(columns=['Unnamed: 0'])

# convert categorical column 'Private' to numerical
college_df['Private'] = college_df['Private'].map({'Yes': 1, 'No': 0})

# define predictors and target variable
X = college_df.drop(columns=['Apps'])
y = college_df['Apps']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=37)

# scale the predictors
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# convert scaled data back to DataFrame
X_train_scaled_college_df = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test_scaled_college_df = pd.DataFrame(X_test_scaled, columns=X.columns)
```

Question 1b

```
# fit a linear regression model
lm = LinearRegression()
lm.fit(X_train_scaled, y_train)

# predict on the test set
y_pred = lm.predict(X_test_scaled)
test_mse = mean_squared_error(y_test, y_pred)
test_mse
```

```
1222954.0382534422
```

- Test MSE: 1222954.0382534422

Question 1c

```
# define 10-fold cross-validation
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# perform PCA on training data
pca = PCA()
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# find the optimal number of principal components
mse_list = []
components_range = range(1, X_train_scaled.shape[1] + 1)

for m in components_range:
    X_train_pca_m = X_train_pca[:, :m]
    lm_pcr = LinearRegression()
    mse = -np.mean(cross_val_score(lm_pcr, X_train_pca_m,
                   y_train, cv=kf, scoring='neg_mean_squared_error'))
    mse_list.append(mse)

optimal_m = components_range[np.argmin(mse_list)]

# fit PCR model using optimal number of components
X_train_pca_opt = X_train_pca[:, :optimal_m]
X_test_pca_opt = X_test_pca[:, :optimal_m]
lm_pcr_opt = LinearRegression()
lm_pcr_opt.fit(X_train_pca_opt, y_train)
y_pcr_pred = lm_pcr_opt.predict(X_test_pca_opt)

# compute MSE
test_mse_pcr = mean_squared_error(y_test, y_pcr_pred)

# determine the elbow point using the "elbow method"
second_derivative = np.diff(mse_list, 2)
elbow_m = components_range[np.argmin(second_derivative) + 1]

optimal_m, test_mse_pcr, elbow_m
```

```
(16, 1487304.1279126806, 3)
```

Question 1d

```
# select the optimal number of components for PLS
mse_list_pls = []

for m in components_range:
    pls = PLSRegression(n_components=m)
    mse = -np.mean(cross_val_score(pls, X_train_scaled, y_train,
                   cv=kf, scoring='neg_mean_squared_error'))
    mse_list_pls.append(mse)

# minimizing cross-validated error
optimal_m_pls = components_range[np.argmin(mse_list_pls)]

# fit PLS model
pls_opt = PLSRegression(n_components=optimal_m_pls)
pls_opt.fit(X_train_scaled, y_train)
y_pls_pred = pls_opt.predict(X_test_scaled)

# compute MSE
test_mse_pls = mean_squared_error(y_test, y_pls_pred)

# determine the elbow point
second_derivative_pls = np.diff(mse_list_pls, 2)
elbow_m_pls = components_range[np.argmin(second_derivative_pls) + 1]

optimal_m_pls, test_mse_pls, elbow_m_pls
```

```
(9, 1300987.697197346, 4)
```

Question 1e

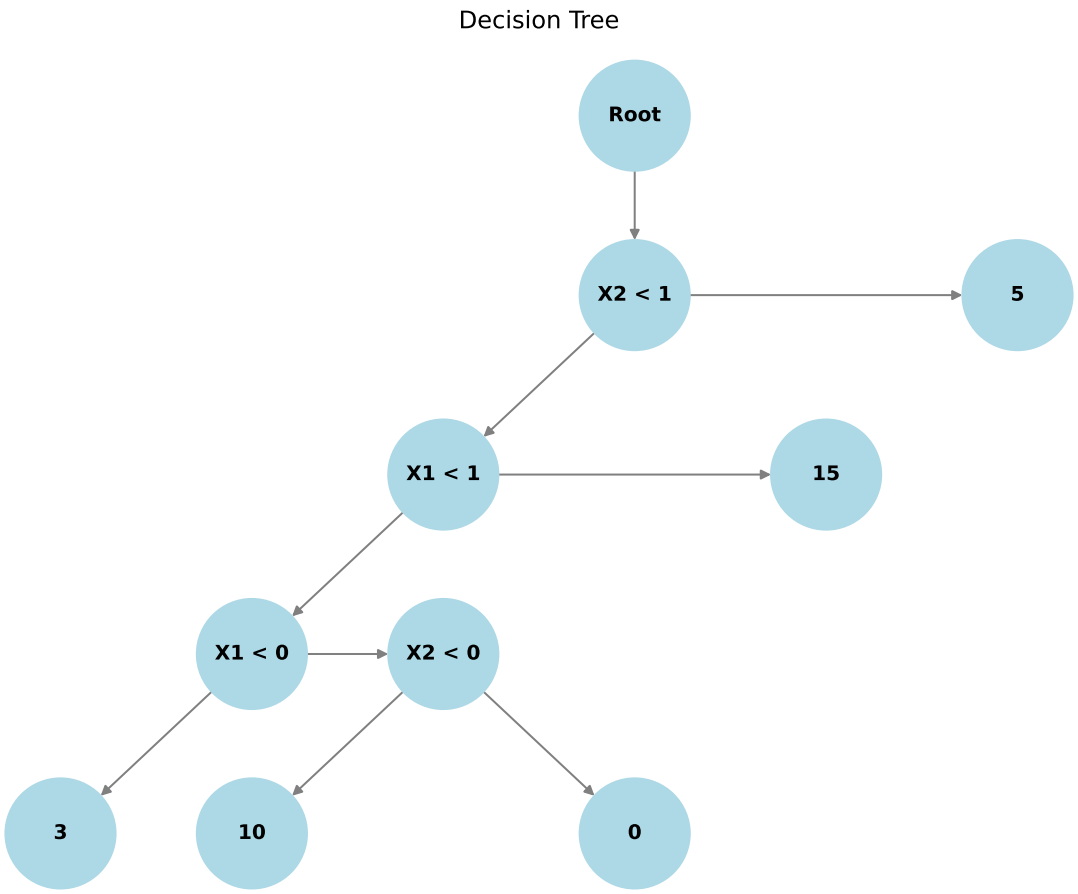
- The linear regression model (OLS) performed best with the lowest test error (1,222,954.04), indicating that all predictors provided valuable information and there was no significant multicollinearity problem. The PLS model (1,300,987.70) performed better than the PCR (1,487,304.13) because it took into account the information of the response variable during the dimensionality reduction process. Both the PCR and PLS models showed high errors, suggesting that the dimensionality reduction process may have lost some important predictive information. Overall, the OLS model remained the most effective prediction tool, while the PLS model provided a good balance between model complexity and prediction performance.

Question 2

Question 2a

```
# reate a decision tree graph
G = nx.DiGraph()

# adding nodes
G.add_node("Root")
G.add_node("X2 < 1")
G.add_node("X1 < 1")
G.add_node("X1 < 0")
G.add_node("X2 < 0")
G.add_node("3")
G.add_node("10")
G.add_node("0")
G.add_node("15")
G.add_node("5")

# adding edges
G.add_edges_from([
    ("Root", "X2 < 1"), ("X2 < 1", "X1 < 1"), ("X1 < 1",
                                                "X1 < 0"), ("X1 < 0", "3"),
                                                 ↪  ("X1 < 0", "X2 < 0"),
    ("X2 < 0", "10"), ("X2 < 0", "0"), ("X1 < 1", "15"), ("X2 < 1", "5")
])

# define positions
pos = {
    "Root": (2, 4),
    "X2 < 1": (2, 3),
    "X1 < 1": (1, 2),
    "X1 < 0": (0, 1),
    "X2 < 0": (1, 1),
    "3": (-1, 0),
    "10": (0, 0),
    "0": (2, 0),
    "15": (3, 2),
    "5": (4, 3)
}
```

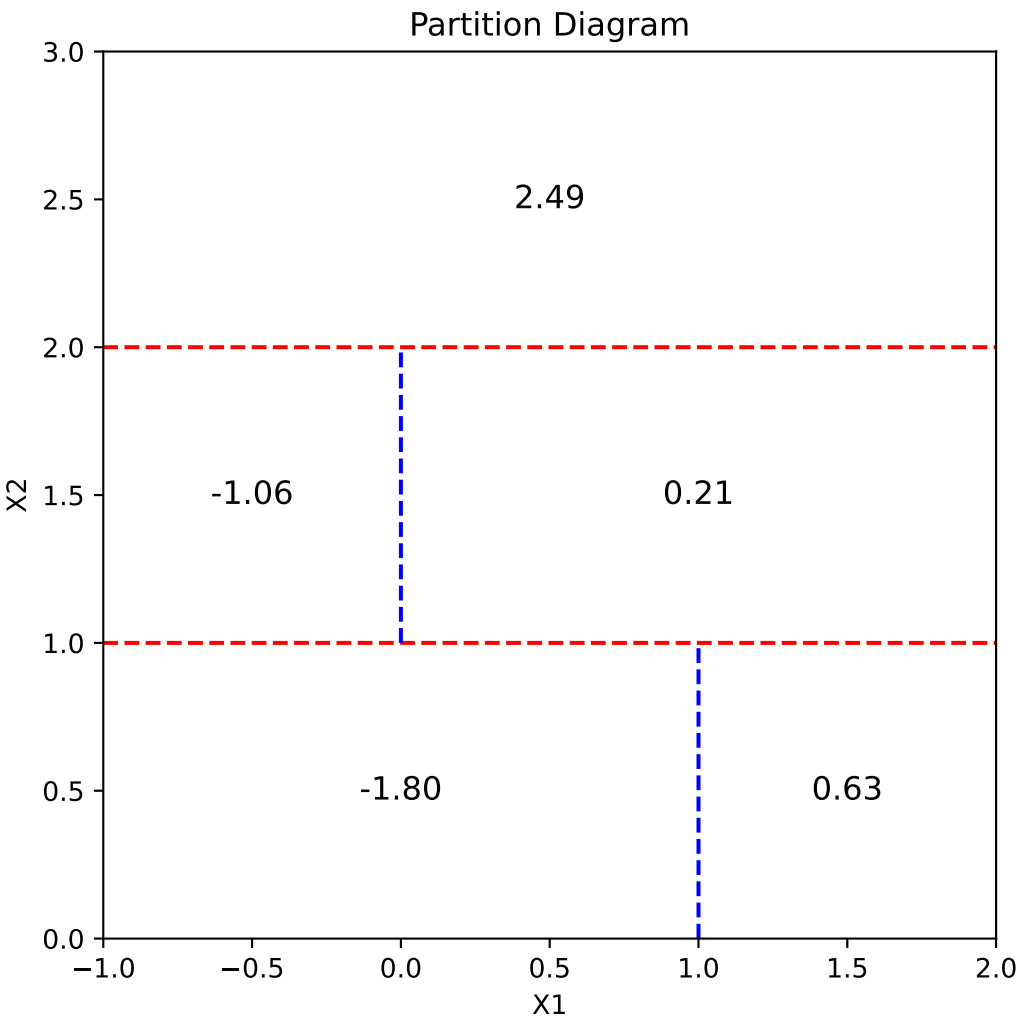
```
# display the graph
plt.figure(figsize=(8, 6))
nx.draw(G, pos, with_labels=True, node_size=3000, node_color="lightblue",
        font_size=10, font_weight="bold", edge_color="gray")
plt.title("Decision Tree")
plt.show()
```



Question 2b

```
fig, ax = plt.subplots(figsize=(6, 6))

# set limits and labels
ax.set_xlim(-1, 2)
```

```
ax.set_ylim(0, 3)
ax.set_xlabel("X1")
ax.set_ylabel("X2")

# draw partition lines
ax.axhline(y=1, color="red", linestyle="dashed")
ax.axhline(y=2, color="red", linestyle="dashed")
ax.vlines(x=1, ymin=0, ymax=1, color="blue", linestyle="dashed")
ax.vlines(x=0, ymin=1, ymax=2, color="blue", linestyle="dashed")

# add region labels (mean values from the tree)
ax.text(0, 0.5, "-1.80", fontsize=12, ha='center', va='center')
ax.text(1.5, 0.5, "0.63", fontsize=12, ha='center', va='center')
ax.text(-0.5, 1.5, "-1.06", fontsize=12, ha='center', va='center')
ax.text(1, 1.5, "0.21", fontsize=12, ha='center', va='center')
ax.text(0.5, 2.5, "2.49", fontsize=12, ha='center', va='center')

# display the plot
plt.title("Partition Diagram")
plt.show()
```



Question 3

```
oj_file_path = "/Users/kevinxu/Documents/GitHub/ML-PS4/Data-OJ.csv"
oj_df = pd.read_csv(oj_file_path)
```

Question 3a

```
# define variable
X = oj_df.drop(columns=['Purchase'])
y = oj_df['Purchase']

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=3)
```

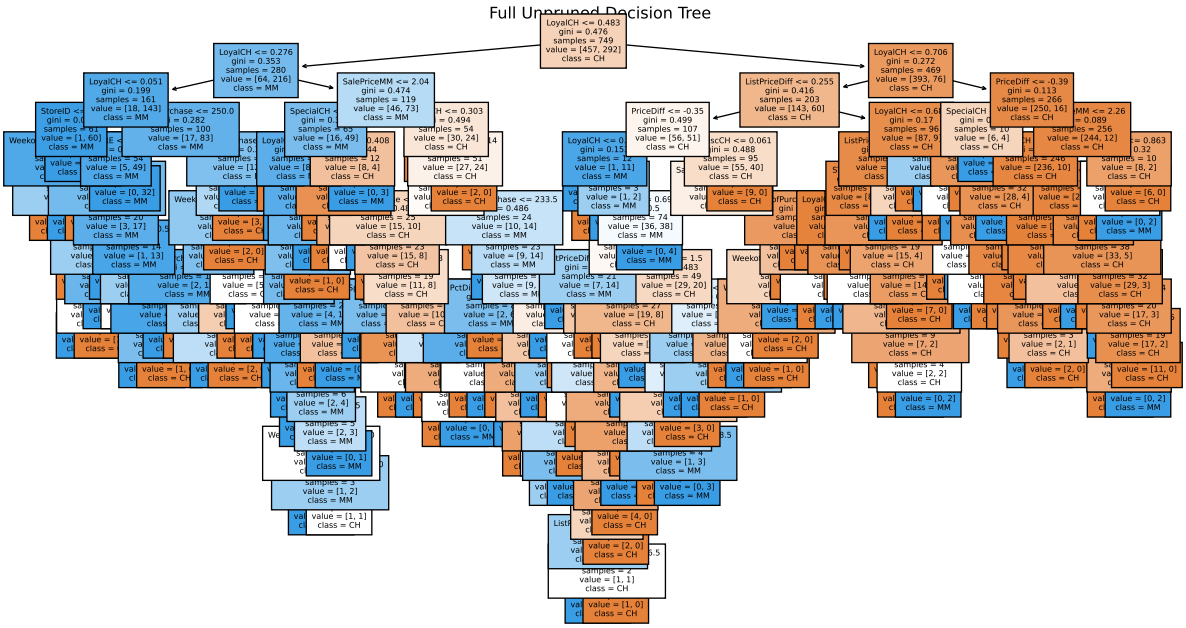
Question 3b

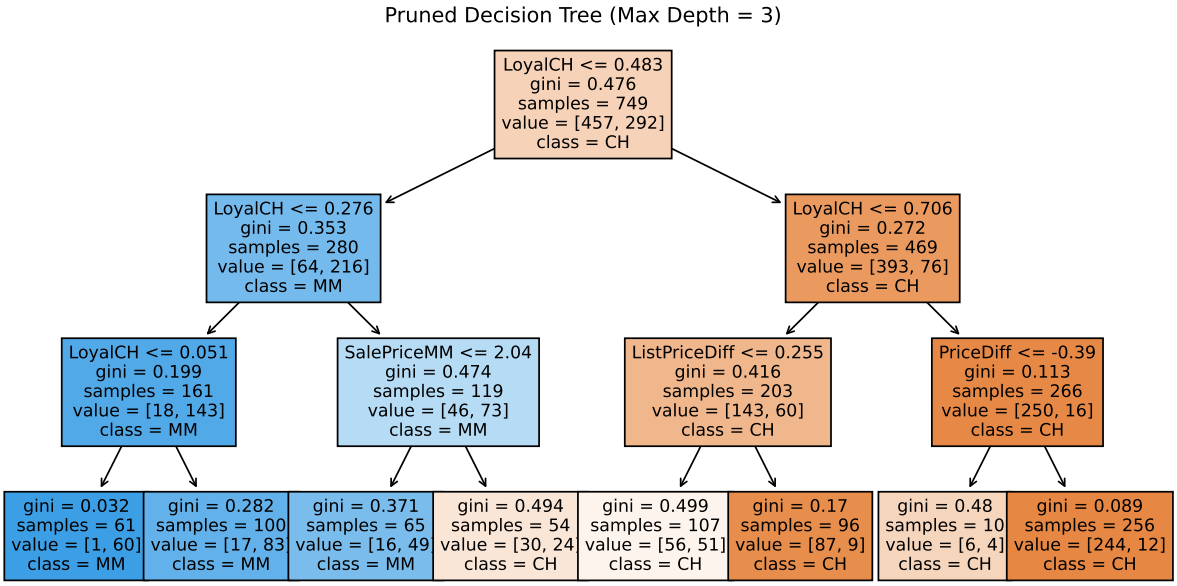
```
# encode the target variable
y_train_encoded = LabelEncoder().fit_transform(y_train)
y_test_encoded = LabelEncoder().fit_transform(y_test)

# convert categorical predictors to numeric values
X_train_encoded = pd.get_dummies(X_train, drop_first=True)
X_test_encoded = pd.get_dummies(X_test, drop_first=True)

X_train_encoded, X_test_encoded = X_train_encoded.align(
    X_test_encoded, join='left', axis=1, fill_value=0)

# fit the full decision tree model
tree_clf = DecisionTreeClassifier(random_state=2)
tree_clf.fit(X_train_encoded, y_train_encoded)

# predict on the training set
y_train_pred = tree_clf.predict(X_train_encoded)

# compute and display the training error rate
train_error_rate = 1 - accuracy_score(y_train_encoded, y_train_pred)
train_error_rate
```

```
0.006675567423230944
```

Question 3c

```
# plot the full, unpruned tree
plt.figure(figsize=(15, 8))
plot_tree(tree_clf, feature_names=X_train_encoded.columns,
          class_names=['CH', 'MM'], filled=True, fontsize=6)
plt.title("Full Unpruned Decision Tree")
plt.show()

# fit a pruned decision tree
tree_clf_pruned = DecisionTreeClassifier(max_depth=3, random_state=2)
tree_clf_pruned.fit(X_train_encoded, y_train_encoded)

# plot the pruned tree
plt.figure(figsize=(12, 6))
plot_tree(tree_clf_pruned, feature_names=X_train_encoded.columns,
          class_names=['CH', 'MM'], filled=True, fontsize=10)
plt.title("Pruned Decision Tree (Max Depth = 3)")
plt.show()

# count the number of terminal nodes
num_terminal_nodes = sum(tree_clf_pruned.tree_.children_left == -1)

num_terminal_nodes
```





8

Question 3d

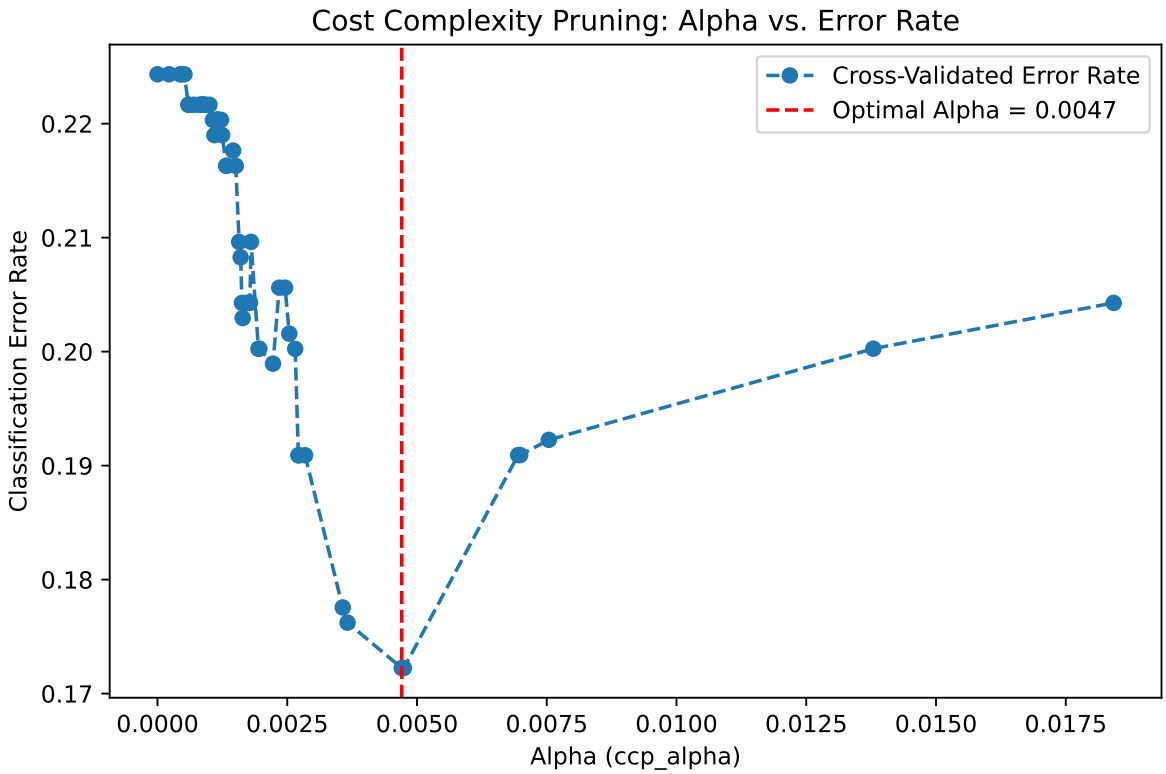
```
# predict on the test set and compute the confusion matrix
y_test_pred = tree_clf.predict(X_test_encoded)
conf_matrix = confusion_matrix(y_test_encoded, y_test_pred)

# display the test error rate
test_error_rate = 1 - accuracy_score(y_test_encoded, y_test_pred)
conf_matrix, test_error_rate
```

```
(array([[160,  36],
        [ 41,  84]]),
 0.23987538940809972)
```

Question 3e

```
# perform CCP path to get alpha values
path = tree_clf.cost_complexity_pruning_path(X_train_encoded,
 ↪  y_train_encoded)
```

```
ccp_alphas = path.ccp_alphas[:-1]

# Perform 5-fold cross-validation
cv_scores = []

for alpha in ccp_alphas:
    pruned_tree = DecisionTreeClassifier(ccp_alpha=alpha, random_state=2)
    scores = cross_val_score(pruned_tree, X_train_encoded,
                             y_train_encoded, cv=5, scoring='accuracy')
    cv_scores.append(1 - np.mean(scores))

# find the optimal alpha
optimal_alpha = ccp_alphas[np.argmin(cv_scores)]

# display the plot
plt.figure(figsize=(8, 5))
plt.plot(ccp_alphas, cv_scores, marker="o", linestyle="dashed",
         label="Cross-Validated Error Rate")
plt.xlabel("Alpha (ccp_alpha)")
plt.ylabel("Classification Error Rate")
plt.title("Cost Complexity Pruning: Alpha vs. Error Rate")
plt.axvline(optimal_alpha, color="red", linestyle="dashed",
            label=f"Optimal Alpha = {optimal_alpha:.4f}")
plt.legend()
plt.show()

optimal_alpha
```



0.0047063975958152315

- The optimal value of corresponding to the lowest cross-validated classification error rate is 0.0047.

Question 3f

```
# compute the tree size for each alpha
tree_sizes = []

for alpha in ccp_alphas:
    pruned_tree = DecisionTreeClassifier(ccp_alpha=alpha, random_state=2)
    pruned_tree.fit(X_train_encoded, y_train_encoded)
    tree_sizes.append(pruned_tree.tree_.node_count)

# find the optimal tree size corresponding to the lowest classification error
optimal_tree_size = tree_sizes[np.argmin(cv_scores)]
```

```
# display the plot
plt.figure(figsize=(8, 5))
plt.plot(tree_sizes, cv_scores, marker="o", linestyle="dashed",
         label="Cross-Validated Error Rate")
plt.xlabel("Tree Size (Number of Nodes)")
plt.ylabel("Classification Error Rate")
plt.title("Tree Size vs. Classification Error Rate")
plt.axvline(optimal_tree_size, color="red", linestyle="dashed",
            label=f"Optimal Tree Size = {optimal_tree_size}")
plt.legend()
plt.show()

optimal_tree_size
```

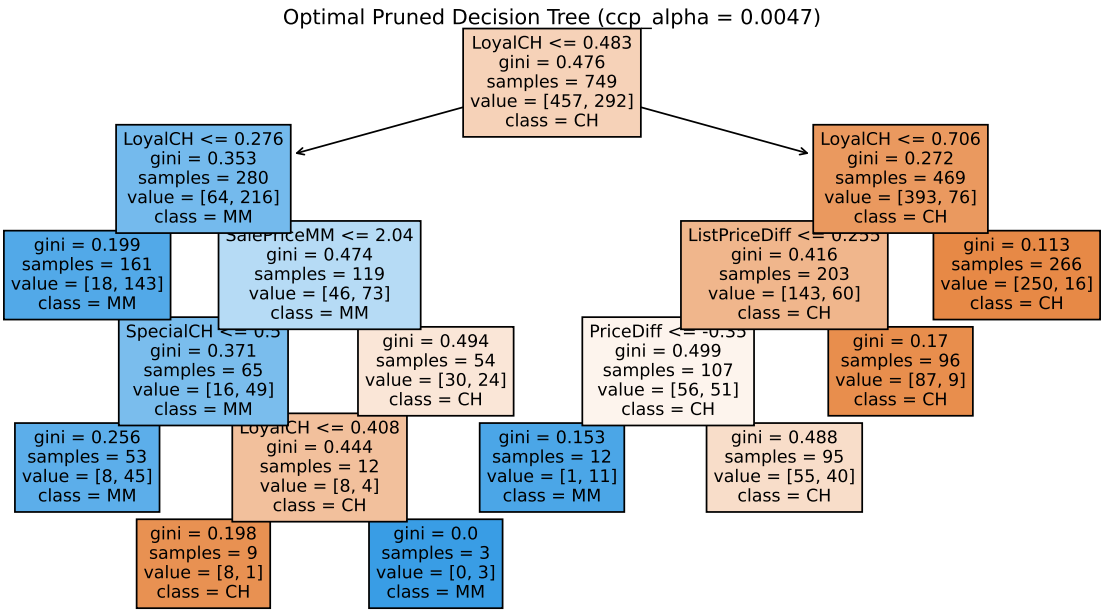
17

- The optimal tree size corresponding to the lowest cross-validated classification error rate is 17 nodes.

- The (ccp_alpha) parameter controls pruning by penalizing the complexity of the tree - higher values result in more aggressive pruning and smaller tree size. Smaller allows for deeper trees, which reduces training error but can easily lead to overfitting and increase test error. Larger simplifies the tree structure and improves generalization but can lead to underfitting. The ideal value should balance model complexity and accuracy, preventing overfitting and underfitting, thereby minimizing classification error.

Question 3g

```
# fit the pruned decision tree using the optimal alpha
optimal_pruned_tree = DecisionTreeClassifier(
    ccp_alpha=optimal_alpha, random_state=2)
optimal_pruned_tree.fit(X_train_encoded, y_train_encoded)

# display the pruned tree
plt.figure(figsize=(12, 6))
plot_tree(optimal_pruned_tree, feature_names=X_train_encoded.columns,
          class_names=['CH', 'MM'], filled=True, fontsize=10)
plt.title(f"Optimal Pruned Decision Tree (ccp_alpha = {optimal_alpha:.4f})")
plt.show()
```



Question 3h

```
y_train_pred_pruned = optimal_pruned_tree.predict(X_train_encoded)
train_error_rate_pruned = 1 - \
    accuracy_score(y_train_encoded, y_train_pred_pruned)

# compare training error rates
train_error_rate, train_error_rate_pruned
```

```
(0.006675567423230944, 0.1562082777036048)
```

- The training error rate for the unpruned tree is 0.67%; the training error rate for the pruned tree is 15.62%.
- Pruning reduces the tree's complexity by removing branches that capture minor variations in the training data, making it less flexible. The unpruned tree overfits by memorizing the training data, resulting in an artificially low training error but a higher test error.

Question 3i

```
y_test_pred_pruned = optimal_pruned_tree.predict(X_test_encoded)
test_error_rate_pruned = 1 - accuracy_score(y_test_encoded,
 ↪  y_test_pred_pruned)

# compare test error rates
test_error_rate, test_error_rate_pruned
```

```
(0.23987538940809972, 0.19626168224299068)
```

- The test error rate for the unpruned tree is 24.3%; the test error rate for the pruned tree is 19.63%. -Pruning removes overly specific splits that capture noise in the training data, preventing overfitting. As a result, the pruned tree generalizes unseen data better, reducing test errors. The unpruned tree overfits the training set, leading to a higher test error due to poor generalization.