HI Decision Tree

October 8, 2024

1 Health Insurance: Decision Tree

1.0.1 Import and Load Data

[8]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import numpy as np
```

[10]:
```
df = pd.read_csv('cleaned_data.csv')
```

[12]:
```
df.head(5)
```

[12]:

	DentalPlan	IssuerID_2014	MultistatePlan_2014	ChildAdultOnly_2014	\
0	1	21989	0	0	
1	1	21989	0	0	
2	1	21989	0	0	
3	1	21989	0	0	
4	1	21989	0	0	

	IssuerID_2015	MultistatePlan_2015	IssuerID_AgeOff2015
0	21989	0	0
1	21989	0	0
2	21989	0	0
3	21989	0	0
4	21989	0	0

[14]:
```
df2 = pd.read_csv('Crosswalk2015.csv')
```

```
/tmp/ipykernel_3018/429338174.py:1: DtypeWarning: Columns (15) have mixed types.
Specify dtype option on import or set low_memory=False.
  df2 = pd.read_csv('Crosswalk2015.csv')
```

[16]:
```
df2.head(4)
```

[16]:
```
  State DentalPlan     PlanID_2014  IssuerID_2014 MultistatePlan_2014  \
0    AK          Y  21989AK0010001          21989                   N
1    AK          Y  21989AK0010001          21989                   N
2    AK          Y  21989AK0010001          21989                   N
3    AK          Y  21989AK0010001          21989                   N

  MetalLevel_2014  ChildAdultOnly_2014  FIPSCode  ZipCode  CrosswalkLevel  \
0             Low                    0      2013        0               1
1             Low                    0      2016        0               1
2             Low                    0      2020        0               1
3             Low                    0      2050        0               1

   …     PlanID_2015 IssuerID_2015  MultistatePlan_2015 MetalLevel_2015  \
0  …  21989AK0030001         21989                    N            High
1  …  21989AK0030001         21989                    N            High
2  …  21989AK0030001         21989                    N            High
3  …  21989AK0030001         21989                    N            High

  ChildAdultOnly_2015 AgeOffPlanID_2015 IssuerID_AgeOff2015  \
0                   0    00000XX0000000                   0
1                   0    00000XX0000000                   0
2                   0    00000XX0000000                   0
3                   0    00000XX0000000                   0

   MultistatePlan_AgeOff2015 MetalLevel_AgeOff2015 ChildAdultOnly_AgeOff2015
0                          X                    X                         X
1                          X                    X                         X
2                          X                    X                         X
3                          X                    X                         X

[4 rows x 21 columns]
```

[18]:
```
df2['MetalLevel_2014'].value_counts(sort=True, ascending=True)
```

[18]:
```
MetalLevel_2014
Platinum         3691
Catastrophic     6464
Gold            21052
High            23465
Bronze          23491
Silver          26931
Low             27411
Name: count, dtype: int64
```

1.1 Encoding

[20]:
```
mapping = {
    'Platinum': 0,
    'Catastrophic': 1,
    'Gold': 2,
    'High': 3,
    'Bronze': 4,
    'Silver': 5,
    'Low': 6
}

df2['MetalLevel_2014'] = df2['MetalLevel_2014'].map(mapping)
```

[22]:
```
mapping = {
    'Platinum': 0,
    'Catastrophic': 1,
    'Gold': 2,
    'High': 3,
    'Bronze': 4,
    'Silver': 5,
    'Low': 6
}

df2['MetalLevel_2015'] = df2['MetalLevel_2015'].map(mapping)
```

[24]:
```
numeric_df2 = df2.select_dtypes(include=['number'])
```

[26]:
```
combined_numeric_df = pd.concat([df, numeric_df2], ignore_index=True)
```

[28]:
```
combined_numeric_df_filled = combined_numeric_df.fillna(0)
```

[30]:
```
combined_numeric_df_filled
```

[30]:

	DentalPlan	IssuerID_2014	MultistatePlan_2014	ChildAdultOnly_2014	\
0	1.0	21989	0.0	0	
1	1.0	21989	0.0	0	
2	1.0	21989	0.0	0	
3	1.0	21989	0.0	0	
4	1.0	21989	0.0	0	
…	…	…	…	…	
265005	0.0	83964	0.0	0	
265006	0.0	83964	0.0	0	
265007	0.0	83964	0.0	0	
265008	0.0	83964	0.0	0	
265009	0.0	83964	0.0	0	

```
        IssuerID_2015  MultistatePlan_2015  IssuerID_AgeOff2015  \
0               21989                  0.0                    0
1               21989                  0.0                    0
2               21989                  0.0                    0
3               21989                  0.0                    0
4               21989                  0.0                    0
…                 …                    …                    …
265005          83964                  0.0                    0
265006          83964                  0.0                    0
265007          83964                  0.0                    0
265008          83964                  0.0                    0
265009          83964                  0.0                    0

        MetalLevel_2014  FIPSCode  ZipCode  CrosswalkLevel  \
0                   0.0       0.0      0.0             0.0
1                   0.0       0.0      0.0             0.0
2                   0.0       0.0      0.0             0.0
3                   0.0       0.0      0.0             0.0
4                   0.0       0.0      0.0             0.0
…                 …         …        …               …
265005              3.0   56037.0      0.0             0.0
265006              3.0   56039.0      0.0             0.0
265007              3.0   56041.0      0.0             0.0
265008              3.0   56043.0      0.0             0.0
265009              3.0   56045.0      0.0             0.0

        ReasonForCrosswalk  MetalLevel_2015
0                      0.0              0.0
1                      0.0              0.0
2                      0.0              0.0
3                      0.0              0.0
4                      0.0              0.0
…                    …                …
265005                 0.0              3.0
265006                 0.0              3.0
265007                 0.0              3.0
265008                 0.0              3.0
265009                 0.0              3.0

[265010 rows x 13 columns]
```

[32]:
```
df = combined_numeric_df
```

1.2 Preprocessing

[34]:
```
df.drop(columns=['IssuerID_2014', 'IssuerID_2015', 'ZipCode'], inplace=True)
```

1.3 Training Model

[36]:
```
X = df.drop('DentalPlan', axis=1)
y = df['DentalPlan']
```

[38]:
```
import numpy as np
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

X_cleaned = X.fillna(X.mean())
y.fillna(y.mode()[0], inplace=True)

X_train, X_val, y_train, y_val = train_test_split(X_cleaned, y, test_size=0.2,␣
 ↪random_state=42)

# Step 1: Define the model
dt = DecisionTreeClassifier(random_state=0)

# Step 2: Define hyperparameters
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 4, 6, 8, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 5, 10],
    'ccp_alpha': np.arange(0.0, 0.1, 0.01)
}

# Step 3: Set up GridSearchCV
grid_search = GridSearchCV(estimator=dt, param_grid=param_grid,
                           cv=5, n_jobs=-1,␣
 ↪scoring='accuracy',return_train_score=True)

# Step 4: Fit the model
grid_search.fit(X, y)

# Step 5: Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)

# Step 6: Evaluate the model with cross-validation
best_model = grid_search.best_estimator_
```

```
cv_scores = cross_val_score(best_model, X_cleaned, y, cv=5)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))

# Step 7: Fit the best model and calculate accuracy
best_model.fit(X_cleaned, y)
y_pred = best_model.predict(X_cleaned)
accuracy = accuracy_score(y, y_pred)
print("Accuracy on the training data:", accuracy)
```

```
Best Parameters: {'ccp_alpha': 0.0, 'criterion': 'gini', 'max_depth': None,
'min_samples_leaf': 1, 'min_samples_split': 2}
Best Cross-Validation Score: 0.6394362476887665
Cross-Validation Scores: [0.834044   0.84855288 0.87249538 0.88983435
0.89479642]
Mean Cross-Validation Score: 0.8679446058639296
Accuracy on the training data: 0.8808724199086827
```

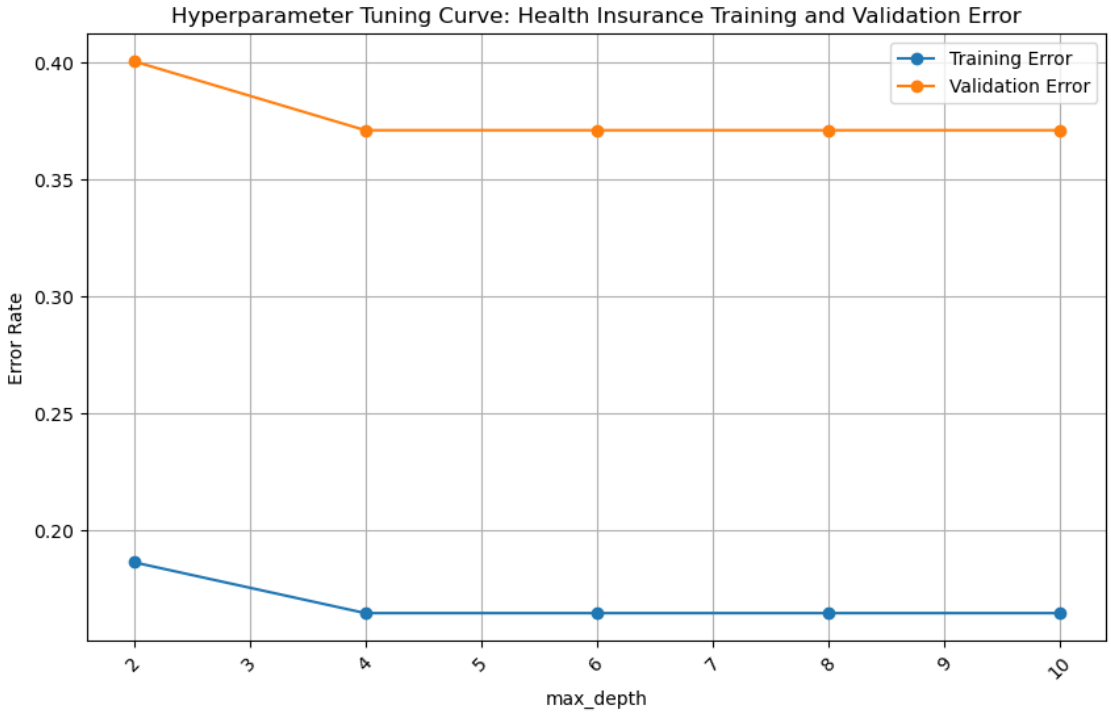
[37]:
```
best_model.fit(X_train, y_train)
y_pred_val = best_model.predict(X_val)
print(classification_report(y_val, y_pred_val))
```

```
              precision    recall  f1-score   support

         0.0       0.87      1.00      0.93     42936
         1.0       0.99      0.38      0.55     10066

    accuracy                           0.88     53002
   macro avg       0.93      0.69      0.74     53002
weighted avg       0.90      0.88      0.86     53002
```

1.4 Training and Validation Error Curve

[39]:
```
results = grid_search.cv_results_

# Extract mean training and validation scores
mean_train_scores = results['mean_train_score']
mean_test_scores = results['mean_test_score']
param_values = results['param_max_depth'].data

# Convert to NumPy arrays for easier processing
param_values = np.array(param_values, dtype=object)
mean_train_scores = np.array(mean_train_scores)
mean_test_scores = np.array(mean_test_scores)
```

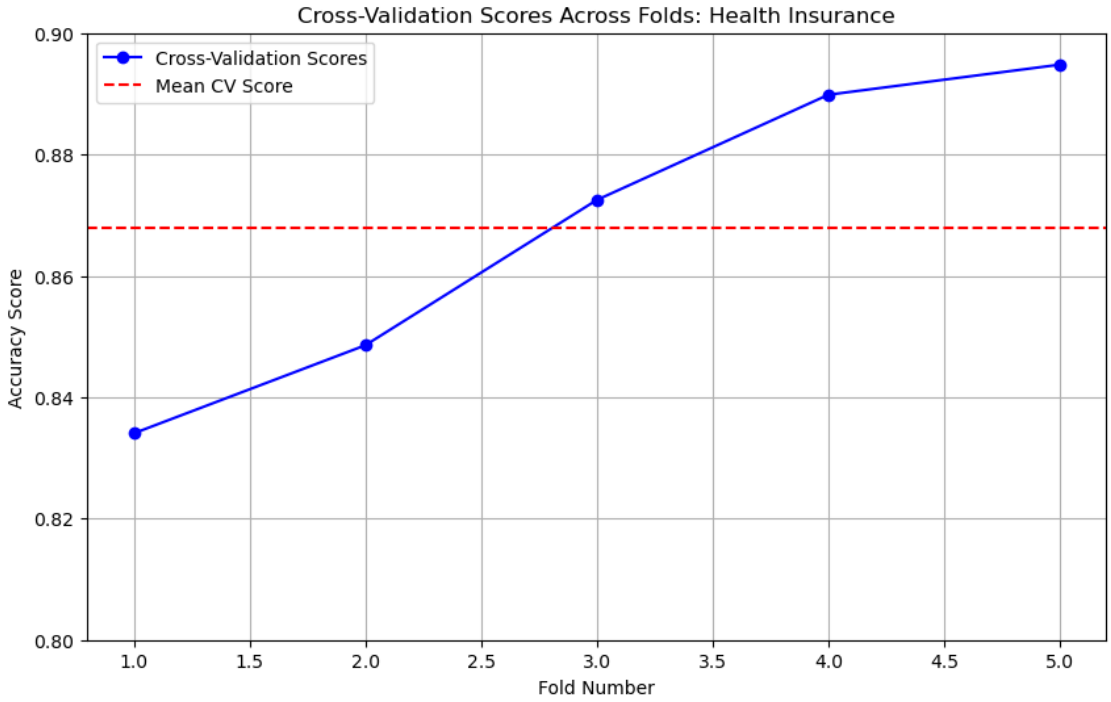
```
results_df = pd.DataFrame({
    'max_depth': param_values,
    'mean_train_score': mean_train_scores,
    'mean_test_score': mean_test_scores
})

# Group by max_depth and calculate mean scores
aggregated_results = results_df.groupby('max_depth').agg({
    'mean_train_score': 'mean',
    'mean_test_score': 'mean'
}).reset_index()

# Calculate errors
aggregated_results['train_error'] = 1 - aggregated_results['mean_train_score']
aggregated_results['validation_error'] = 1 -␣
 ↪aggregated_results['mean_test_score']

# Plotting the aggregated errors
plt.figure(figsize=(10, 6))
plt.plot(aggregated_results['max_depth'], aggregated_results['train_error'],␣
 ↪marker='o', label='Training Error')
plt.plot(aggregated_results['max_depth'],␣
 ↪aggregated_results['validation_error'], marker='o', label='Validation Error')
plt.title("Hyperparameter Tuning Curve: Health Insurance Training and␣
 ↪Validation Error")
plt.xlabel("max_depth")
plt.ylabel("Error Rate")
plt.xticks(rotation=45)
plt.legend()
plt.grid()
plt.show()
```



1.5 Cross Validation Scores Across Folds

[41]:
```
best_params = {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1,␣
↪'min_samples_split': 2}
best_cv_score = 0.6394362476887665
cv_scores = [0.834044, 0.84855288, 0.87249538, 0.88983435, 0.89479642]
mean_cv_score = 0.8679446058639296
training_accuracy = 0.8808724199086827
```

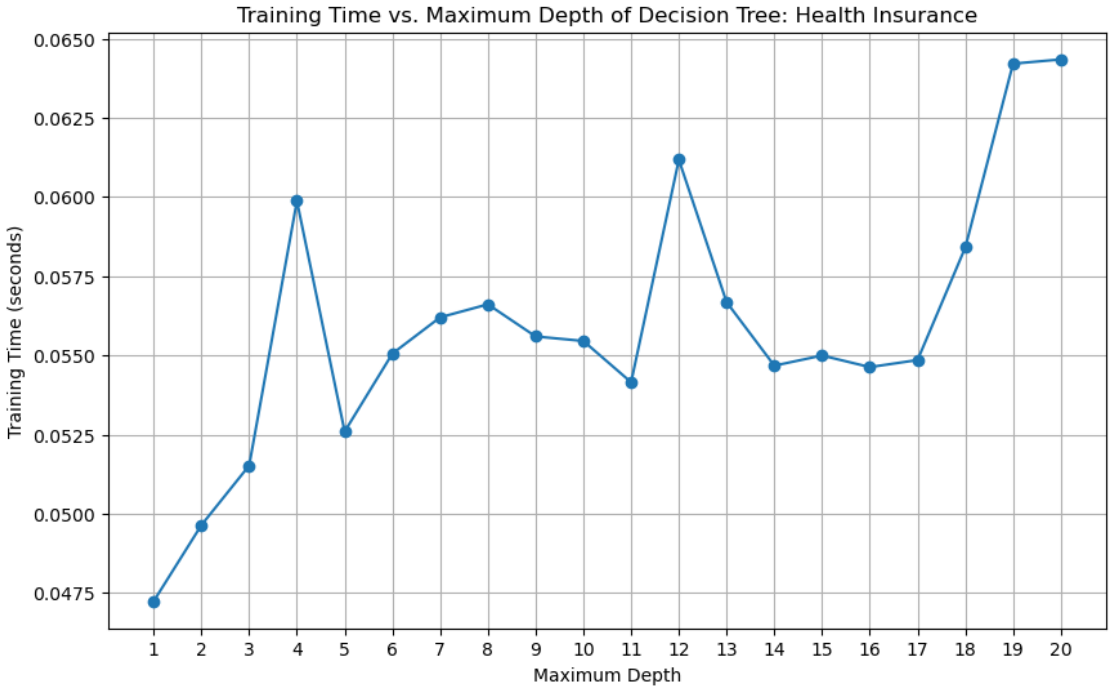
[43]:
```
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cv_scores) + 1), cv_scores, marker='o', linestyle='-',␣
↪color='blue', label='Cross-Validation Scores')
plt.axhline(y=mean_cv_score, color='red', linestyle='--', label='Mean CV Score')
plt.title('Cross-Validation Scores Across Folds: Health Insurance')
plt.xlabel('Fold Number')
plt.ylabel('Accuracy Score')
plt.ylim(0.8, 0.9)
plt.legend()
plt.grid(True)
plt.show()
```



1.6 Training Time

[66]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import time


max_depths = np.arange(1, 21)
training_times = []

# Measure training time for different max depths
for max_depth in max_depths:
    clf = DecisionTreeClassifier(max_depth=max_depth)

    start_time = time.time()
    clf.fit(X_train, y_train)
    end_time = time.time()

    training_times.append(end_time - start_time)

# Plotting the training time vs. max depth
```

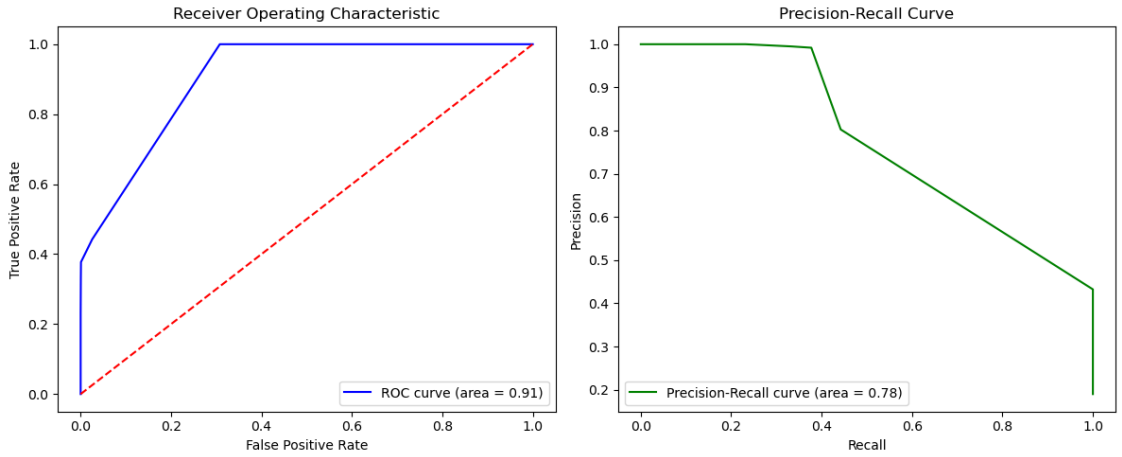
```
plt.figure(figsize=(10, 6))
plt.plot(max_depths, training_times, marker='o')
plt.title('Training Time vs. Maximum Depth of Decision Tree: Health Insurance')
plt.xlabel('Maximum Depth')
plt.ylabel('Training Time (seconds)')
plt.xticks(max_depths)
plt.grid()
plt.show()
```



1.7 ROC Curve

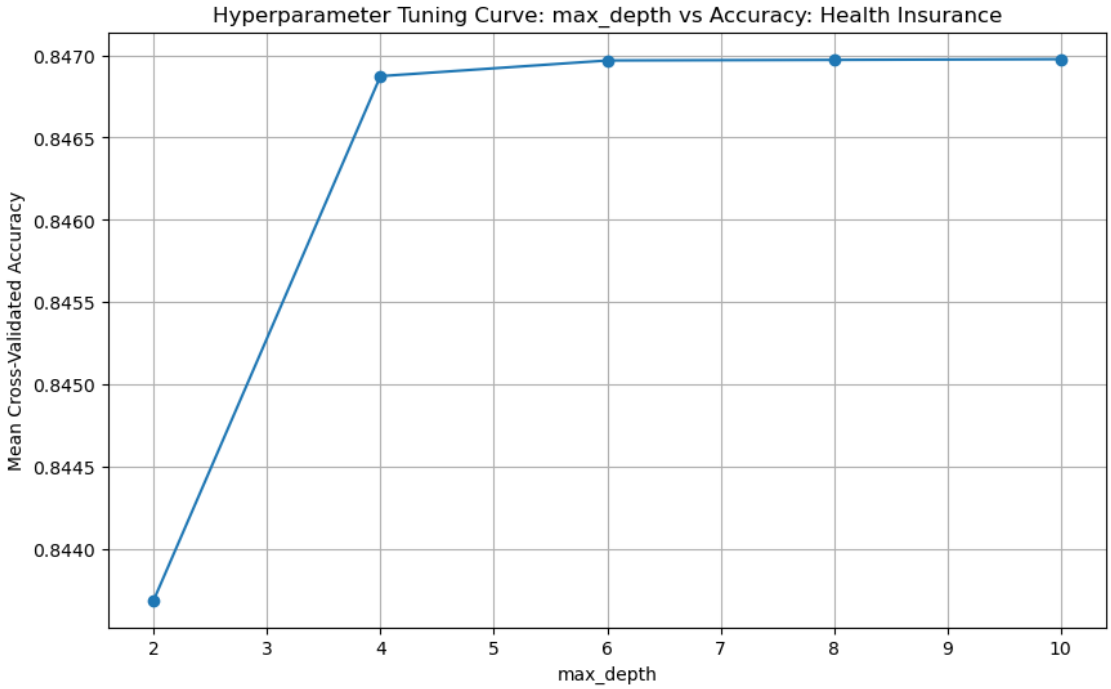
[45]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve, auc, precision_recall_curve


# Train the Decision Tree classifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Get predicted probabilities
```

```
y_probs = clf.predict_proba(X_val)[:, 1]

# ROC Curve
fpr, tpr, _ = roc_curve(y_val, y_probs)
roc_auc = auc(fpr, tpr)

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_val, y_probs)
pr_auc = auc(recall, precision)

# Plotting
plt.figure(figsize=(12, 5))

# ROC Curve
plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('Receiver Operating Characteristic')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')

# Precision-Recall Curve
plt.subplot(1, 2, 2)
plt.plot(recall, precision, color='green', label=f'Precision-Recall curve (area␣
 ↪= {pr_auc:.2f})')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='lower left')

plt.tight_layout()
plt.show()
```



1.8 Hyperparameter Curve: Max_depth against Accuracy

[47]:
```
import matplotlib.pyplot as plt

grid_search.fit(X_train, y_train)

# Get the results
results = grid_search.cv_results_

# Extract mean test scores and parameters
mean_test_scores = results['mean_test_score']
param_values = results['param_max_depth'].data

# Convert to numpy arrays
param_values = np.array(param_values)
mean_test_scores = np.array(mean_test_scores)

# Filter out None values and corresponding test scores
valid_indices = ~np.isnan(mean_test_scores) & (param_values != None)
filtered_param_values = param_values[valid_indices]
filtered_mean_test_scores = mean_test_scores[valid_indices]


results_df = pd.DataFrame({
    'max_depth': filtered_param_values,
    'mean_test_score': filtered_mean_test_scores
})

# Group by max_depth and calculate the mean scores
grouped_results = results_df.groupby('max_depth').mean().reset_index()

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(grouped_results['max_depth'], grouped_results['mean_test_score'],␣
 ↪marker='o')
plt.title("Hyperparameter Tuning Curve: max_depth vs Accuracy: Health␣
 ↪Insurance")
plt.xlabel("max_depth")
plt.ylabel("Mean Cross-Validated Accuracy")
plt.grid()
plt.show()
```


Hyperparameter Tuning Curve: max_depth vs Accuracy: Health Insurance
Mean Cross-Validated Accuracy
0.8470
0.8465
0.8460
0.8455
0.8450
0.8445
0.8440
2
3
4
5
6
7
8
9
10
max_depth

[ ]: