In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from IPython.display import Image
```

## Prepare the data set for Phishing vs Benign

In [2]:
```
# Load data into a dataframe named "df"
df = pd.read_csv('DataSetForPhishingVSBenignUrl.csv', header=0)
df
```

Out[2]:

| | Querylength | domain_token_count | path_token_count | avgdomaintokenlen | longdomaintok |
|---|---|---|---|---|---|
| **0** | 0 | 4 | 5 | 5.500000 | |
| **1** | 0 | 4 | 5 | 5.500000 | |
| **2** | 0 | 4 | 5 | 5.500000 | |
| **3** | 0 | 4 | 12 | 5.500000 | |
| **4** | 0 | 4 | 6 | 5.500000 | |
| **...** | ... | ... | ... | ... | |
| **36702** | 29 | 4 | 14 | 5.750000 | |
| **36703** | 0 | 4 | 13 | 3.750000 | |
| **36704** | 58 | 3 | 27 | 6.666666 | |
| **36705** | 35 | 3 | 13 | 4.333334 | |
| **36706** | 40 | 3 | 25 | 6.666666 | |

36707 rows × 80 columns

In [3]:
```
# Print the shape of the original data set, for reference
print(f"shape of data set before dropping na rows: {df.shape}")

# Drop any rows that have NaN values
df.dropna(inplace=True)

# Print the shape of the modified data set
print(f"shape of data set after dropping na rows: {df.shape}")
```

```
shape of data set before dropping na rows: (36707, 80)
shape of data set after dropping na rows: (18982, 80)
```

In [4]:
```
# Pull out only the "benign" and "phishing" records. Then combine into
benign_data = df[df['URL_Type_obf_Type'] == "benign"]
phishing_data = df[df['URL_Type_obf_Type'] == "phishing"]

df = pd.concat([benign_data, phishing_data])
print(f"shape of data set with only benign and phishing: {df.shape}")
```

```
shape of data set with only benign and phishing: (6723, 80)
```

In [5]:
```
# Set the feature set (X) to all columns of the data set except the la
X = df.drop(columns=df.columns[-1],
        axis=1,
        inplace=False)
print(f"shape of X: {X.shape}")

# Create an ndarray "y" from the last column of the data set.
y = df.iloc[:, -1].values
print(f"shape of y: {y.shape}")
print()

# Print the number of records of each class
from collections import Counter
class_counts = Counter(y)
print("Count of records of each type:")
for c in class_counts:
    print(f"{c}: {class_counts[c]}")
```

```
shape of X: (6723, 79)
shape of y: (6723,)

Count of records of each type:
benign: 2709
phishing: 4014
```

In [6]:
```
# Split the data into a training set (80%) and testing set (20%).
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
print(f"training set shape: {X_train.shape}")
print(f"testing set shape: {X_test.shape}")
```

```
training set shape: (5378, 79)
testing set shape: (1345, 79)
```

## Run the model for trees of depth 1, 2, 3, 4, 5, and 6 and for the Gini Impurity and Entropy impurity measures for each tree depth. Compare the results of these 12 cases and discuss your results.

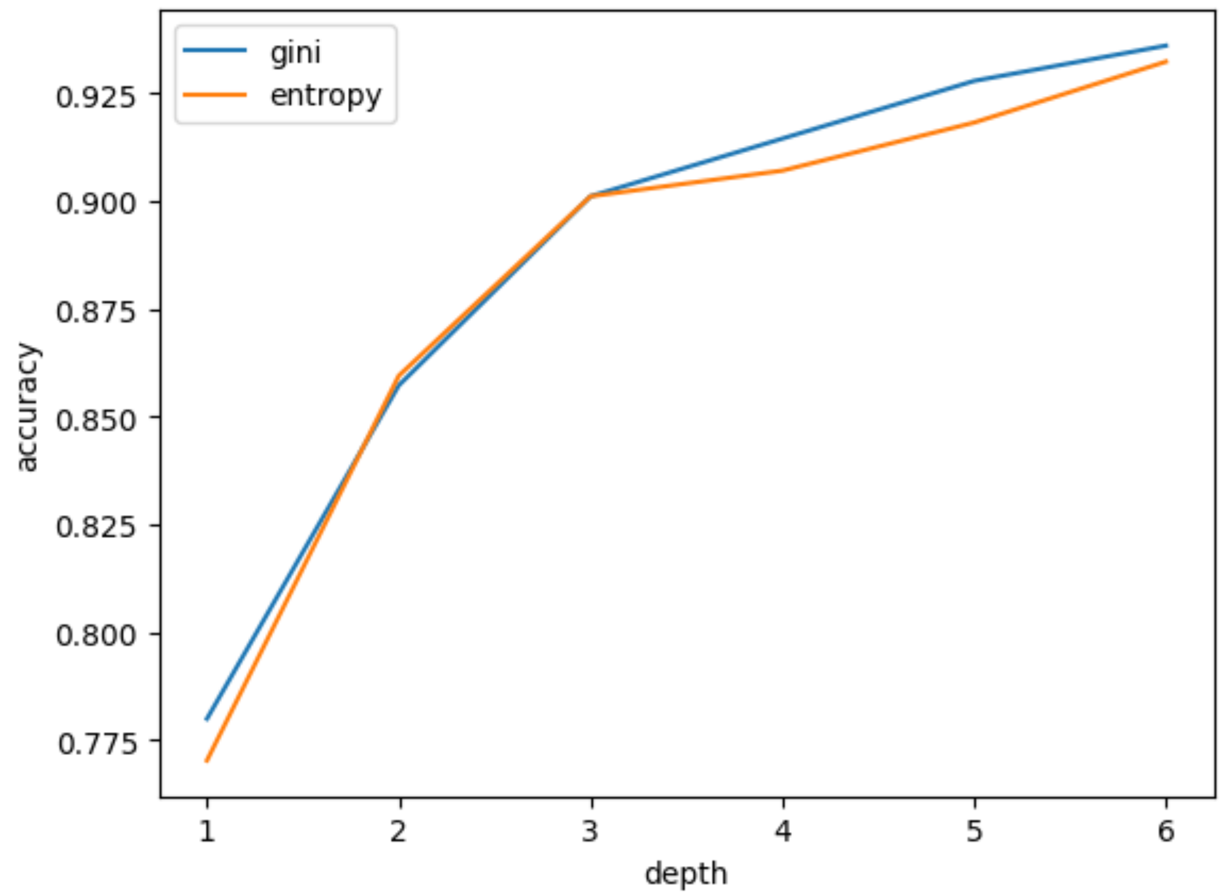
In [7]:
```
# Train a decision tree classifier and generate results for each of th
scores = {"gini": [], "entropy": []}
for d in range(1,7):
    for crit in ("gini", "entropy"):
        clf = DecisionTreeClassifier(random_state=0, max_depth=d, crit
        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)
        scores[crit].append(score) # Add current score to scores
        print(f"criterion: {crit:8}  depth: {d}  accuracy: {score:.4f}
```

```
criterion: gini      depth: 1  accuracy: 0.7799
criterion: entropy   depth: 1  accuracy: 0.7703
criterion: gini      depth: 2  accuracy: 0.8572
criterion: entropy   depth: 2  accuracy: 0.8595
criterion: gini      depth: 3  accuracy: 0.9011
criterion: entropy   depth: 3  accuracy: 0.9011
criterion: gini      depth: 4  accuracy: 0.9145
criterion: entropy   depth: 4  accuracy: 0.9071
criterion: gini      depth: 5  accuracy: 0.9279
criterion: entropy   depth: 5  accuracy: 0.9182
criterion: gini      depth: 6  accuracy: 0.9361
criterion: entropy   depth: 6  accuracy: 0.9323
```

In [8]:
```
%matplotlib inline

# Create a graph, where the X-axis is the depth and Y-axis is the accu
# each criterion (gini and entropy).
depths = range(1,7)
gini = np.array(scores["gini"])
entropy = np.array(scores["entropy"])

plt.plot(depths, gini, label='gini')
plt.plot(depths, entropy, label='entropy')
plt.ylabel('accuracy')
plt.xlabel('depth')
plt.legend(loc='upper left')
plt.show()
```




As seen above, the accuracy of the decision tree increases as the depth of the decision tree increases. The accuracy has begun to level off by depth 6; any more depth is likely to result in overfitting. There is not much difference between the accuracy for gini versus entropy. At depth of 2, the accuracy score for entropy is slightly higher, so we will use it for the visualization in the next step.

# Take the best performing tree of depth 2 from above. Visualize the tree and discuss your observations.

In [17]:
```
# Create decision tree classifier with depth=2, criterion=entropy
clf = DecisionTreeClassifier(random_state=0, max_depth=2, criterion =
clf.fit(X,y)
```

Out[17]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state
=0)
```

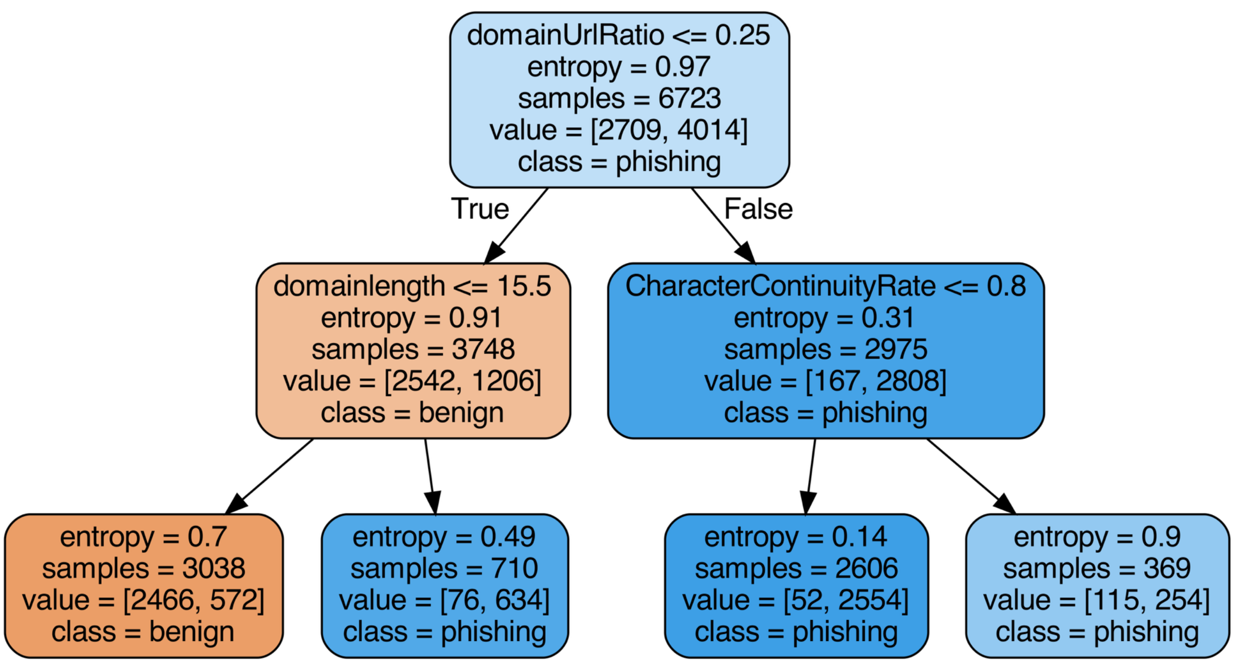
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [18]:
```
# Export graph to .dot file
export_graphviz(clf, out_file='tree.dot', feature_names = X.columns,
                class_names = np.unique(y),
                rounded = True, proportion = False, precision = 2, fil
```

In [19]:
```
# Convert .dot file to .png file
!dot -Tpng tree.dot -o tree.png -Gdpi=600
```

In [20]:
```
# Display .png file
Image(filename = 'tree.png')
```

Out[20]:



The decision tree starts by splitting on the feature "domainUrlRatio" with a value of 0.25. We can see that while the original dataset is 60% phishing URL's, those with a "domainUrlRatio" less than 0.25 are primarily benign (2542 of 3748, or 68%). Those with a "domainUrlRatio" of over 0.25 are overwhelmingly Phishing URL's (2808 of 2975, or 94%).

At the second level, the left tree is split on "domainlength" less than or equal to 15.5. This splits the data into a set which is 81% benign (domainlength <= 15.5), and a set which is 89% Phishing URLs (domainlength > 15.5).

The right tree (domainUrlRatio > 0.25) is primarily Phishing URL's, and the decision tree decides to split on CharacterContinuityRate <= 0.8. Both sides are still predominantly Phishing URL's, but the left side is extremely clean (98% Phishing URL's), while the right side is more mixed (69% Phishing URL's).

Overall, we can see the entropy lowering as we move in depth down the tree, from 0.97 at depth 0, to 0.91 and 0.31 at depth 1, to finally 0.7, 0.49, 0.14, and 0.9 at depth 2.

In [ ]: