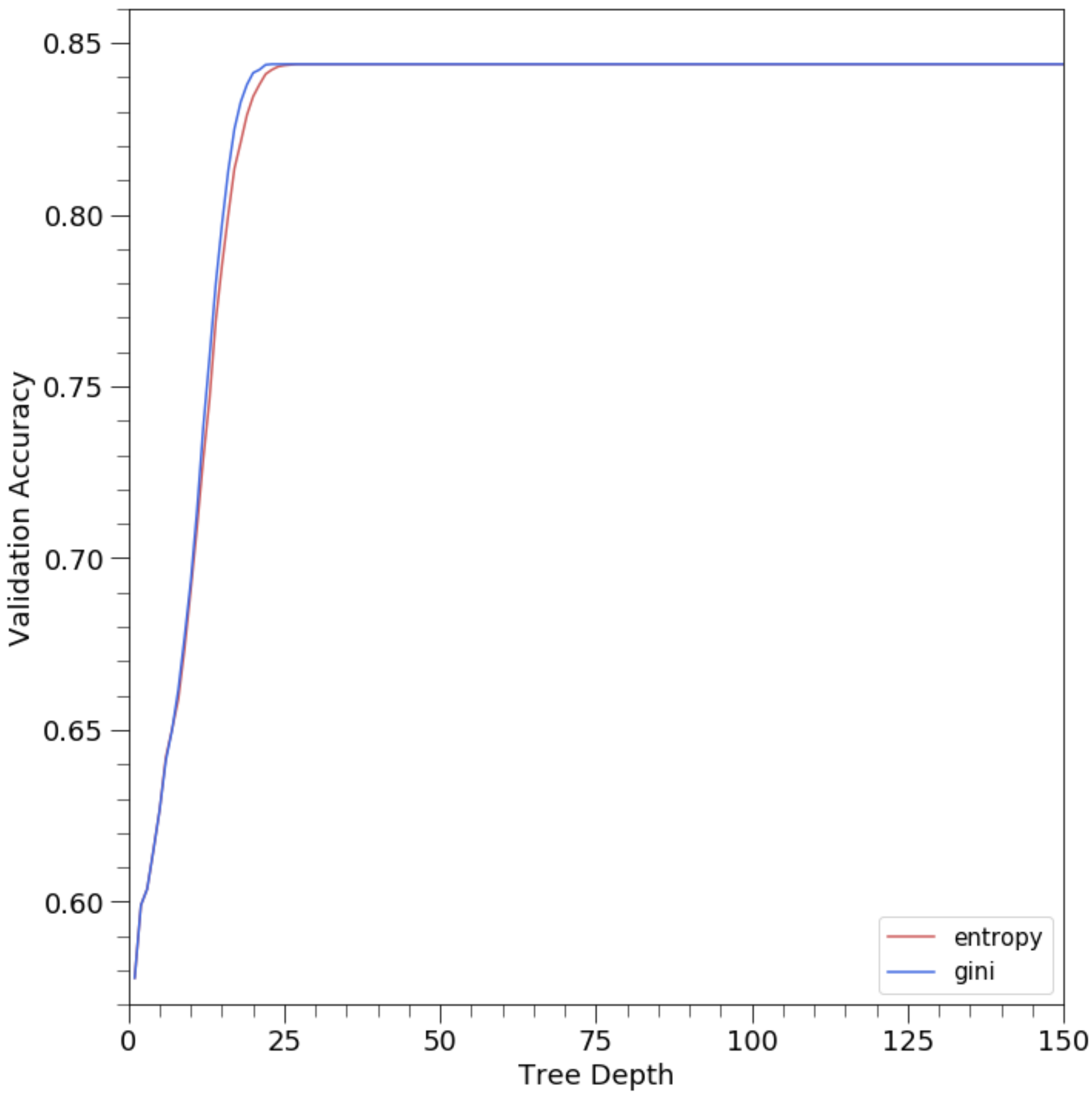
In [7]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Dec  3 21:50:54 2019

@author: jorgeagr
"""

import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

width = 10
height = 10

mpl.rcParams['figure.figsize'] = (width, height)
mpl.rcParams['font.size'] = 18
mpl.rcParams['figure.titlesize'] = 'small'
mpl.rcParams['legend.fontsize'] = 'small'
mpl.rcParams['xtick.major.size'] = 12
mpl.rcParams['xtick.minor.size'] = 8
mpl.rcParams['xtick.labelsize'] = 18
mpl.rcParams['ytick.major.size'] = 12
mpl.rcParams['ytick.minor.size'] = 8
mpl.rcParams['ytick.labelsize'] = 18
```

In [8]:
```
# load training set
data_train = pd.read_csv('../data/election_data_train.csv')

# split into features and labels
x_train = data_train.values[:, :-1]
y_train = data_train.values[:,-1]

# Split the training data in 10 folds to perform cross validation
# and see performance as a function of number of trees
folds = 10
kfold = KFold(n_splits=folds, shuffle=True, random_state=0)
max_trees = np.arange(150) + 1
```

In [9]:
```
# find optimal model
fig, ax = plt.subplots()
criterion_list = ['entropy', 'gini']
colors = ['indianred', 'royalblue']
highest_acc = 0
for c, criterion in enumerate(criterion_list):
    num_tree_accval = np.zeros((len(max_trees), 2))
    num_tree_tpr = np.zeros((len(max_trees), 2))
    num_tree_tnr = np.zeros((len(max_trees), 2))
    for t in max_trees:
        val_acc = np.zeros(folds)
        val_tpr = np.zeros(folds)
        val_tnr = np.zeros(folds)
        for i, inds in enumerate(kfold.split(x_train)):
            train_ind = inds[0]
            val_ind = inds[1]
            tree = DecisionTreeClassifier(criterion=criterion, max_depth=t, rando
m_state=0)
            tree.fit(x_train, y_train)
            val_acc[i] = tree.score(x_train[val_ind], y_train[val_ind])
            # Confusion Matrix considers the case vote = 1 as positive,
            # so considers voting for Rep
            tn, fp, fn, tp = confusion_matrix(y_train[val_ind], tree.predict(x_tr
ain[val_ind])).ravel()
            val_tpr[i] = tp / (tp + fn)
            val_tnr[i] = tn / (tn + fp)
            if val_acc[i] > highest_acc:
                # save the best tree (highest acc)
                best_tree = tree
                highest_acc = val_acc[i]
        num_tree_accval[t-1,0] = val_acc.mean()
        num_tree_accval[t-1,1] = val_acc.std()
        num_tree_tpr[t-1,0] = val_tpr.mean()
        num_tree_tpr[t-1,1] = val_tpr.std()
        num_tree_tnr[t-1,0] = val_tnr.mean()
        num_tree_tnr[t-1,1] = val_tnr.std()

    acc_meanval = num_tree_accval[:,0]
    acc_stdval = num_tree_accval[:,1]
    ax.plot(max_trees, acc_meanval, color=colors[c], label=criterion, zorder=1)
ax.xaxis.set_major_locator(mtick.MultipleLocator(25))
ax.xaxis.set_minor_locator(mtick.MultipleLocator(5))
ax.yaxis.set_major_locator(mtick.MultipleLocator(0.05))
ax.yaxis.set_minor_locator(mtick.MultipleLocator(0.01))
ax.set_xlim(0, 150)
ax.set_ylim(0.57, 0.86)
ax.set_xlabel('Tree Depth')
ax.set_ylabel('Validation Accuracy')
ax.legend(loc='lower right')
fig.tight_layout(pad=0.5)
#fig.savefig('../prob2.eps', dpi=500)

print('Optimal performance with {} criterion, {} trees with {:.2f} +/- {:.2f}% ac
c, {:.2f} +/- {:.2f}% tpr, {:.2f} +/- {:.2f}% tnr'.format(best_tree.criterion,
          best_tree.max_depth,
          num_tree_accval[best_tree.max_depth-1][0]*100, num_tree_accval[best_tre
e.max_depth-1][1]*100,
          num_tree_tpr[best_tree.max_depth-1][0]*100, num_tree_tpr[best_tree.max_
depth-1][1]*100,
          num_tree_tnr[best_tree.max_depth-1][0]*100, num_tree_tnr[best_tree.max_
depth-1][1]*100))
```

```
Optimal performance with entropy criterion, 24 trees with 84.38 +/- 1.30% acc, 80.
48 +/- 1.14% tpr, 87.95 +/- 2.09% tnr
```



In [10]:

```
# load testing set to predict election results
data_test = pd.read_csv('../data/election_data_test.csv')
x_test = data_test.values[:,:-1]
# need to take into account vote weight due to (racial) turnout
w_test = data_test.values[:,-1]

# decided to predict using the different training set folds,
# this way we can obtain an uncertainty in the election results
total_votes = w_test.sum()
dem_results = np.zeros(folds)
rep_results = np.zeros(folds)
for i, inds in enumerate(kfold.split(x_train)):
        train_ind = inds[0]
        tree = DecisionTreeClassifier(criterion='entropy', max_depth=best_tree.ma
x_depth, random_state=0)
        tree.fit(x_train[train_ind], y_train[train_ind])
        y_pred = tree.predict(x_test)
        dem_results[i] = w_test[y_pred==0].sum() / total_votes
        rep_results[i] = w_test[y_pred==1].sum() / total_votes

print('Dem votes: {:.2f} +/- {:.2f}'.format(dem_results.mean()*100, dem_results.s
td()*100))
print('Rep votes: {:.2f} +/- {:.2f}'.format(rep_results.mean()*100, rep_results.s
td()*100))
```

```
Dem votes: 58.78 +/- 3.14
Rep votes: 41.22 +/- 3.14
```

In [ ]: