In [113.
```
import pandas as pd
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [114.
```
diabetes_data = pd.read_csv('C:/Users/sophi/Desktop/UM_bachelor/year_2/Machine Learning/Lab 1/diabetes.csv')
glass_data = pd.read_csv('C:/Users/sophi/Desktop/UM_bachelor/year_2/Machine Learning/Lab 1/glass.csv')
```

In [115.
```
# Y_d is the class column, which stores one of the two values: tested_negative, tested_positive
# X_d is everuthing but class column and id (I guess)

Y_d = diabetes_data['class']
X_d = diabetes_data.drop(['class'],axis=1)

Y_g = glass_data['class']
X_g = glass_data.drop(['class'],axis=1)
```

c. Train one-level decision trees and multi-level decision trees on the two data sets. Determine the accuracy rates of the resulting classifiers using the training set and hold- out validation1. Explain why there is a difference in the accuracy rates. Compare one- level decision trees and multi-level decision trees in terms of explainability.

In [116.
```
# Splits the data into training and test

X_train_diabetes, X_test_diabetes, Y_train_diabetes, Y_test_diabetes = train_test_split(X_d, Y_d,
test_size=0.34, random_state=10)

X_train_glass, X_test_glass, Y_train_glass, Y_test_glass = train_test_split(X_g, Y_g,
test_size=0.34, random_state=10)
```

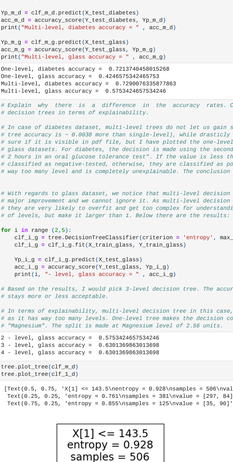
In [117.
```
# Building decision trees

# one-level
clf_1_d = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)
clf_1_g = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = 1)

clf_1_d = clf_1_d.fit(X_train_diabetes, Y_train_diabetes)
clf_1_g = clf_1_g.fit(X_train_glass, Y_train_glass)

# multi-level
clf_m_d = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)
clf_m_g = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = None)

clf_m_d = clf_m_d.fit(X_train_diabetes, Y_train_diabetes)
clf_m_g = clf_m_g.fit(X_train_glass, Y_train_glass)
```

In [118.
```
# Accuracy of the predictions

# one-level

Yp_1_d = clf_1_d.predict(X_test_diabetes)
acc_1_d = accuracy_score(Y_test_diabetes, Yp_1_d)
print("One-level, diabetes accuracy = " , acc_1_d)

Yp_1_g = clf_1_g.predict(X_test_glass)
acc_1_g = accuracy_score(Y_test_glass, Yp_1_g)
print("One-level, glass accuracy = ", acc_1_g)

# multi-level

Yp_m_d = clf_m_d.predict(X_test_diabetes)
acc_m_d = accuracy_score(Y_test_diabetes, Yp_m_d)
print("Multi-level, diabetes accuracy = " , acc_m_d)

Yp_m_g = clf_m_g.predict(X_test_glass)
acc_m_g = accuracy_score(Y_test_glass, Yp_m_g)
print("Multi-level, glass accuracy = " , acc_m_g)

One-level, diabetes accuracy =  0.7213740458015268
One-level, glass accuracy =  0.42465753424657534
Multi-level, diabetes accuracy =  0.7290076335877863
Multi-level, glass accuracy =  0.5753424657534246
```

In [119.
```
# Explain why there is a difference in the accuracy rates. Compare one-level decision trees and multi-level
# decision trees in terms of explainability.

# In case of diabetes dataset, multi-level trees do not let us gain significantly more accuracy (the multi-level
# tree accuracy is ~ 0.0038 more than single-level), while drastically increasing the complexity of explainability. I am not
# sure if it is visible in pdf file, but I have plotted the one-level and multi-level decision tree for both diabetes and
# glass datasets. For diabetes, the decision is made using the second attribute, which is "Plasma glucose concentration a
# 2 hours in an oral glucose tolerance test". If the value is less than or equal to 143.5, then the instances are
# classified as negative-tested, otherwise, they are classified as positive-tested. Taking the multi-level tree, it has
# way too many level and is completely unexplainable. The conclusion here is that one-level decision tree is good enough.


# With regards to glass dataset, we notice that multi-level decision trees increase the accuracy by 25% (!!). This is a
# major improvement and we cannot ignore it. As multi-level decision trees are not limited in the maximum number of steps,
# they are very likely to overfit and get too complex for understanding. Thus, I tried to keep the restriction on max number
# of levels, but make it larger than 1. Below there are the results:

for i in range (2,5):
    clf_i_g = tree.DecisionTreeClassifier(criterion = 'entropy', max_depth = i)
    clf_i_g = clf_i_g.fit(X_train_glass, Y_train_glass)

    Yp_i_g = clf_i_g.predict(X_test_glass)
    acc_i_g = accuracy_score(Y_test_glass, Yp_i_g)
    print(i, "- level, glass accuracy = " , acc_i_g)

# Based on the results, I would pick 3-level decision tree. The accuracy is increase by more than 20% and the complexity
# stays more or less acceptable.

# In terms of explainability, multi-level decision tree in this case, same as for diabetes dataset, is ahrdly explainable,
# as it has way too many levels. One-level tree makes the decision considering the value of the third attribute, which is
# "Magnesium". The split is made at Magnesium level of 2.66 units.

2 - level, glass accuracy =  0.5753424657534246
3 - level, glass accuracy =  0.6301369863013698
4 - level, glass accuracy =  0.6301369863013698
```

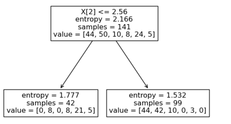
In [120.
```
tree.plot_tree(clf_m_d)
tree.plot_tree(clf_1_d)
```

Out[120]:
```
[Text(0.5, 0.75, 'X[1] <= 143.5\nentropy = 0.928\nsamples = 506\nvalue = [332, 174]'),
 Text(0.25, 0.25, 'entropy = 0.761\nsamples = 381\nvalue = [297, 84]'),
 Text(0.75, 0.25, 'entropy = 0.855\nsamples = 125\nvalue = [35, 90]')]
```



In [121.
```
tree.plot_tree(clf_m_g)
tree.plot_tree(clf_1_g)
```

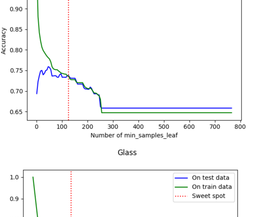
Out[121]:
```
[Text(0.5, 0.75, 'X[2] <= 2.56\nentropy = 2.166\nsamples = 141\nvalue = [44, 50, 10, 8, 24, 5]'),
 Text(0.25, 0.25, 'entropy = 1.777\nsamples = 42\nvalue = [0, 8, 0, 8, 21, 5]'),
 Text(0.75, 0.25, 'entropy = 1.532\nsamples = 99\nvalue = [44, 42, 10, 0, 3, 0]')]
```

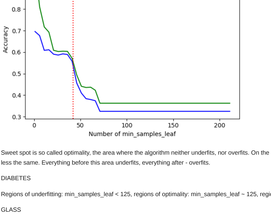


d. Experiment with multi-level decision trees and error pre-pruning by changing the option min_samples_leaf from 0 to the size of the datasets (use some step). (The option min_samples_leaf determines the min number of training instances in the leaf nodes of the decision trees.) Estimate the accuracy rates of the resulting decision trees using the training set and hold-out validation. Plot the accuracy rates based on the training set and hold-out validation for min_samples_leaf from 1 to the size of the datasets with step of 5. Identify the regions of underfitting, optimality, and overfitting. Explain how you have identified these regions.

In [122.
```
run_num = 10
################################################################################
# DIABETES

n = 768
x_d = [0] * round(n/5)
y_d = [0] * round(n/5)
y_d_trainonly = [0] * round(n/5)

for k in range (0, run_num):
    di = 0

    X_train_diabetes, X_test_diabetes, Y_train_diabetes, Y_test_diabetes = train_test_split(X_d, Y_d,
    test_size=0.34, random_state = k)

# X_train_diabetes, X_test_diabetes, Y_train_diabetes, Y_test_diabetes = train_test_split(X_d, Y_d,
# test_size=0.34, random_state = 10)

    for i in range (1, n, 5):

        # Building decision trees, diabetes
        clf_d = tree.DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf = i)
        clf_d = clf_d.fit(X_train_diabetes, Y_train_diabetes)

        # Accuracy of the predictions on train data, diabetes
        Yp_d_train = clf_d.predict(X_train_diabetes)
        acc_d_train = accuracy_score(Y_train_diabetes, Yp_d_train)

        y_d_trainonly[di] += acc_d_train

        # Accuracy of the predictions on test data, diabetes
        Yp_d = clf_d.predict(X_test_diabetes)
        acc_d = accuracy_score(Y_test_diabetes, Yp_d)
        # print("i = ", i, "; Diabetes accuracy = " , acc_d)

        x_d[di] = i
        y_d[di] += acc_d
        di+=1

for k in range (0, len(y_d)):
    y_d[k] /= run_num
    y_d_trainonly[k] /= run_num

################################################################################
# GLASS

n = 214
x_g = [0] * round(n/5)
y_g = [0] * round(n/5)
y_g_trainonly = [0] * round(n/5)
gj = 0

for k in range (0, run_num):
    gj = 0

    X_train_glass, X_test_glass, Y_train_glass, Y_test_glass = train_test_split(X_g, Y_g,
    test_size=0.34, random_state = k)

    for j in range (1, n, 5):

        # Building decision trees, glass
        clf_g = tree.DecisionTreeClassifier(criterion = 'entropy', min_samples_leaf = j)
        clf_g = clf_g.fit(X_train_glass, Y_train_glass)

        # Accuracy of the predictions on train data, glass
        Yp_g_train = clf_g.predict(X_train_glass)
        acc_g_train = accuracy_score(Y_train_glass, Yp_g_train)

        y_g_trainonly[gj] += acc_g_train

        # Accuracy of the predictions on test data, glass
        Yp_g = clf_g.predict(X_test_glass)
        acc_g = accuracy_score(Y_test_glass, Yp_g)
        # print("j = ", j, "; Glass accuracy = " , acc_g)

        x_g[gj] = j
        y_g[gj] += acc_g
        gj+=1

for k in range (0, len(y_g)):
    y_g[k] /= run_num
    y_g_trainonly[k] /= run_num
```

In [123.
```
import matplotlib.pyplot as plt

plt.suptitle('Diabetes')
plt.plot(x_d, y_d, 'b', label = "On test data")
plt.plot(x_d, y_d_trainonly, 'g', label = "On train data")
plt.ylabel('Accuracy')
plt.xlabel('Number of min_samples_leaf')
plt.axvline(x = 125, color = 'r', label = "Sweet spot", linestyle = ':')
leg = plt.legend(loc="upper right")
plt.show()

plt.suptitle('Glass')
plt.plot(x_g, y_g, 'b', label = "On test data")
plt.plot(x_g, y_g_trainonly, 'g', label = "On train data")
plt.ylabel('Accuracy')
plt.xlabel('Number of min_samples_leaf')
plt.axvline(x = 42, color = 'r', label = "Sweet spot", linestyle = ':')
leg = plt.legend(loc="upper right")
plt.show()
```





Sweet spot is so called optimality, the area where the algorithm neither underfits, nor overfits. On the graoh it is the area where the accuracy on test data and train data are more or less the same. Everything before this area underfits, everything after - overfits.

DIABETES

Regions of underfitting: min_samples_leaf < 125, regions of optimality: min_samples_leaf ~ 125, regions of overfitting: min_samples_leaf > 125

GLASS

Regions of underfitting: min_samples_leaf < 42, regions of optimality: min_samples_leaf ~ 42, regions of overfitting: min_samples_leaf > 42