Sudarshan.M.S

CB.EN.U4CSE18258

POML Assignment

In [54]:
```
!pip install -q dtreeviz
```

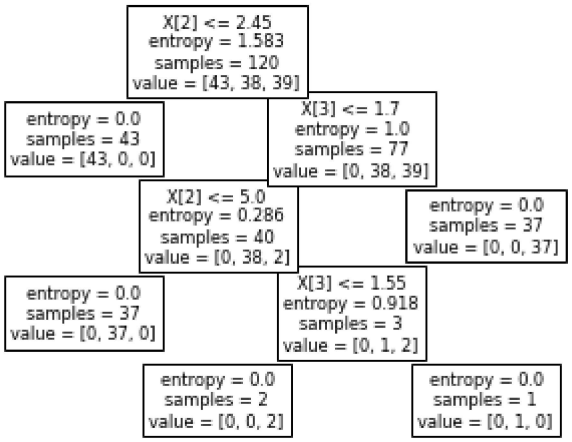
```
     |████████████████████████████████| 51kB 2.5MB/s
  Building wheel for dtreeviz (setup.py) ... done
```

In [85]:
```
import pprint
import graphviz
import numpy as np
import pandas as pd
import seaborn as sns
from dtreeviz.trees import *
from sklearn import tree
from sklearn import metrics
from graphviz import Source
from numpy import log2 as log
from sklearn.tree import export_text
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [86]:
```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=7)
classifier = tree.DecisionTreeClassifier(criterion='entropy')
classifier = classifier.fit(X_train, y_train)
```

In [87]:
```
df=sns.load_dataset("iris")
df_train,df_test = train_test_split(df,train_size=0.8,random_state=7)
```

In [88]:
```
tree.plot_tree(clf);
```



In [89]:
```
iris = load_iris()
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=None)
decision_tree = decision_tree.fit(iris.data, iris.target)
r = export_text(decision_tree, feature_names=iris['feature_names'])
viz1 = dtreeviz(decision_tree,
                X_train,
                y_train,
                target_name='Class',
                feature_names=iris.feature_names,
                class_names=["setosa", "versicolor", "virginica"],
                histtype= 'barstacked')  # barstackes is default
print(r)
```

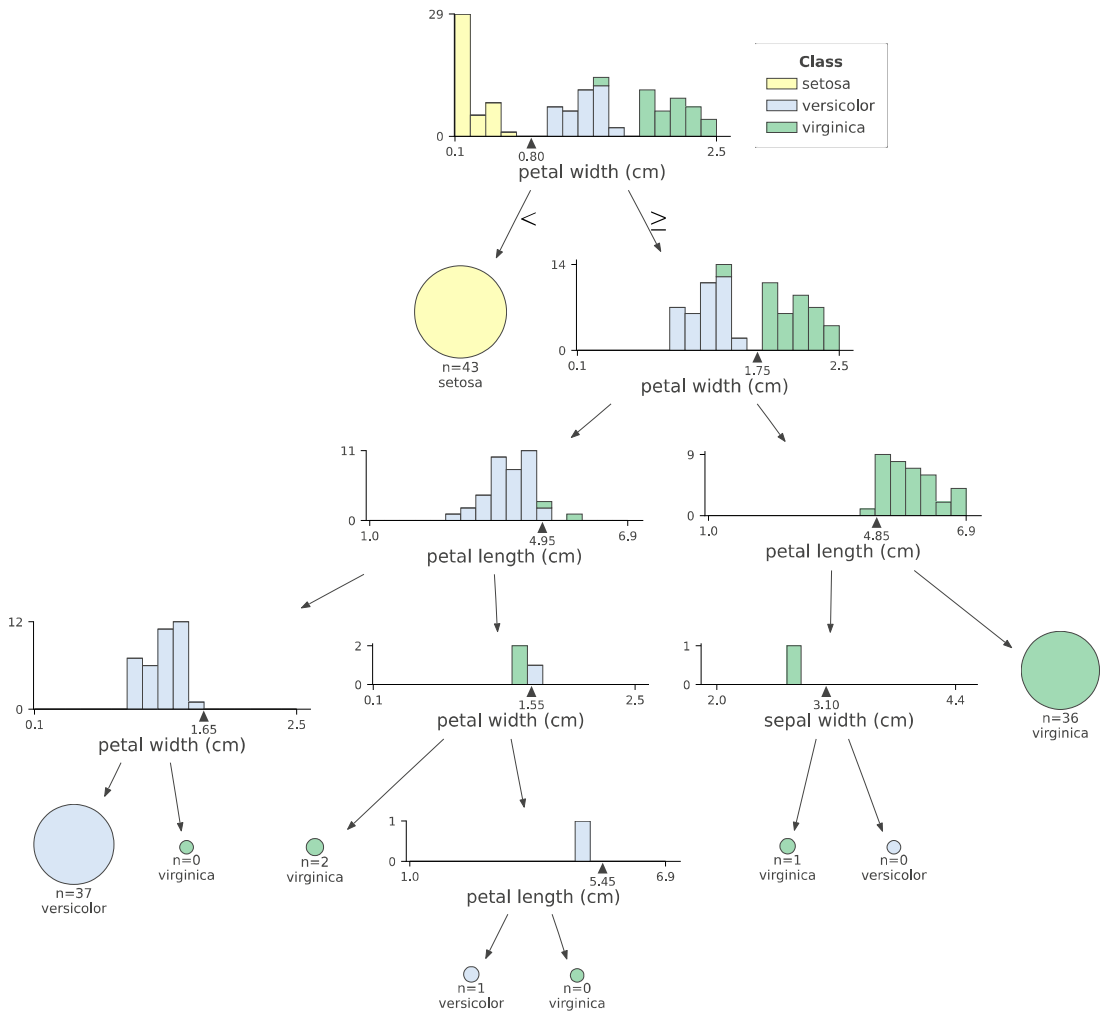
```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDepr
ecationWarning: Creating an ndarray from ragged nested sequences (which is a
list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shape
s) is deprecated. If you meant to do this, you must specify 'dtype=object' wh
en creating the ndarray
  return array(a, dtype, copy=False, order=order)

|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- petal width (cm) <= 1.65
|   |   |   |   |--- class: 1
|   |   |   |--- petal width (cm) >  1.65
|   |   |   |   |--- class: 2
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- petal width (cm) <= 1.55
|   |   |   |   |--- class: 2
|   |   |   |--- petal width (cm) >  1.55
|   |   |   |   |--- petal length (cm) <= 5.45
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- petal length (cm) >  5.45
|   |   |   |   |   |--- class: 2
|   |--- petal width (cm) >  1.75
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

Simple graph without histogram or scatterpots for decision nodes (knowledge dataset)

In [90]:
```
viz1
```

Out[90]:

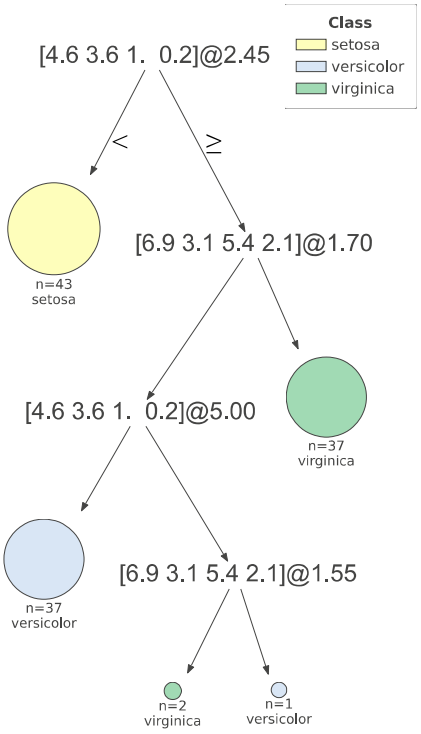


Simple graph without histogram or scatterpots for decision nodes (Iris dataset)

In [91]:
```
viz2 = dtreeviz(clf,
                X_train,
                y_train,
                target_name='Class',
                feature_names=X_train,
                class_names=["setosa", "versicolor", "virginica"],
                fancy=False)
```

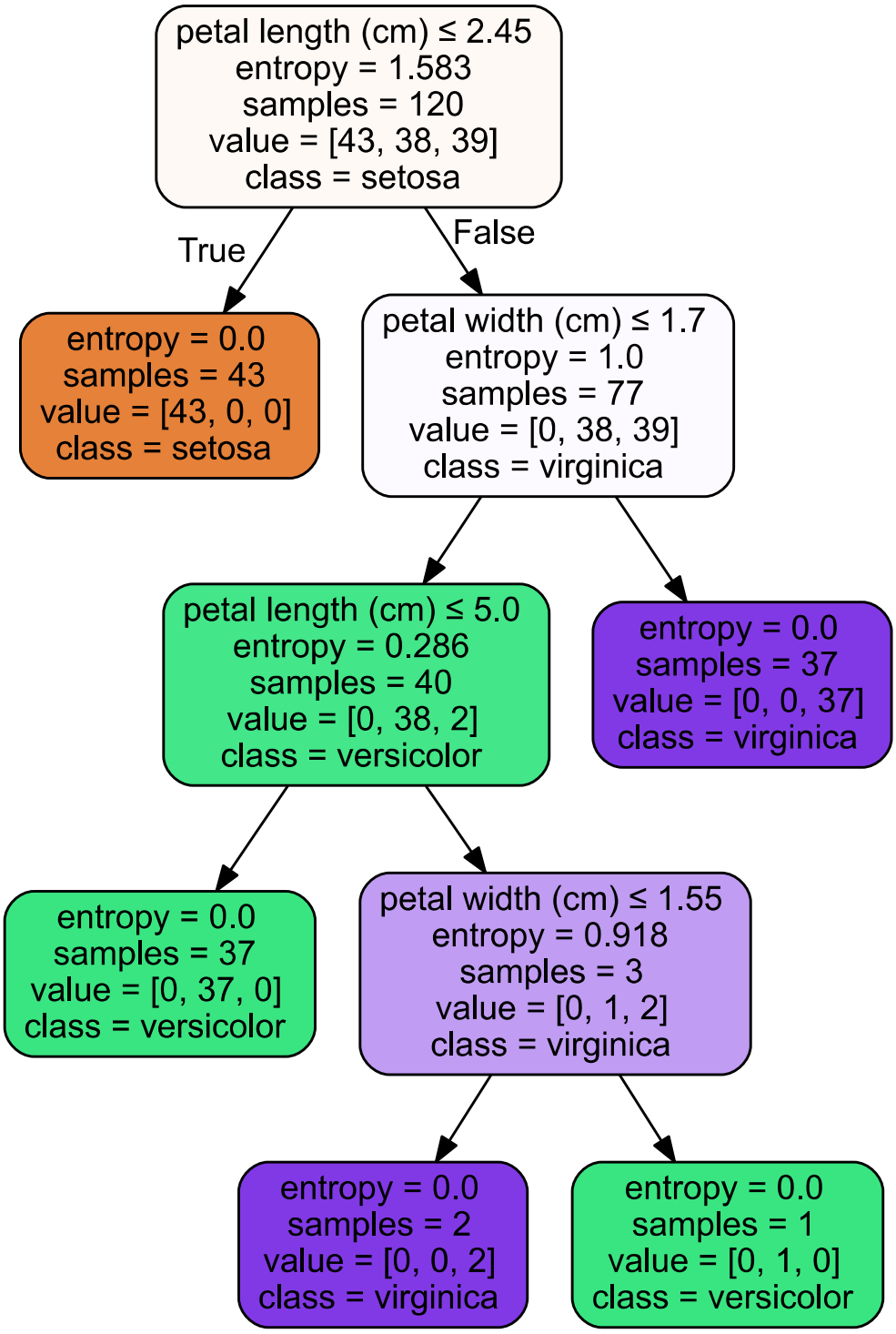
In [92]:
```
viz2
```

Out[92]:



In [93]:
```
dot_data = tree.export_graphviz(clf, out_file=None, feature_names=iris.feature
_names,class_names=iris.target_names,filled=True, rounded=True,special_charact
ers=True)
graph = graphviz.Source(dot_data)
graph
```

Out[93]:



In [94]:
```
prediction = clf.predict(X_test)
clf.score(X_test,y_test)*100
```

Out[94]: 86.66666666666667

In [95]:
```
# eps for making value a bit greater than 0 later on
eps = np.finfo(float).eps
```

In [96]:
```
def find_entropy(df):
    Class = df.keys()[-1]
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy
```

In [97]:
```
def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]
    target_variables = df[Class].unique()
    variables = df[attribute].unique()
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
                num = len(df[attribute][df[attribute]==variable][df[Class]==ta
rget_variable])
                den = len(df[attribute][df[attribute]==variable])
                fraction = num/(den+eps)
                entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)
```

In [98]:
```
def find_winner(df):
    Entropy_att=[]
    IG=[]
    for key in df.keys()[:-1]:
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]
```

In [99]:
```
def get_subtable(df, node, value):
    return df[df[node]==value].reset_index(drop=True)
```

In [100]:
```
def buildTree(df,tree=None):
    Class = df.keys()[-1]
    node = find_winner(df)
    attValue = np.unique(df[node])
    if tree is None:
        tree={}
        tree[node] = {}
    for value in attValue:
        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable[Class],return_counts=True)

        if len(counts)==1:
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable)
    return tree
```

In [101]:
```
tree = buildTree(df_train)
```

In [102]:
```
pprint.pprint(tree)
```

```
{'petal_width': {0.1: 'setosa',
                 0.2: 'setosa',
                 0.3: 'setosa',
                 0.4: 'setosa',
                 0.5: 'setosa',
                 0.6: 'setosa',
                 1.0: 'versicolor',
                 1.1: 'versicolor',
                 1.2: 'versicolor',
                 1.3: 'versicolor',
                 1.4: {'sepal_width': {2.6: 'virginica',
                                       2.7: 'versicolor',
                                       2.8: 'versicolor',
                                       3.0: 'versicolor',
                                       3.1: 'versicolor',
                                       3.2: 'versicolor'}},
                 1.5: {'petal_length': {4.2: 'versicolor',
                                        4.5: 'versicolor',
                                        4.6: 'versicolor',
                                        4.7: 'versicolor',
                                        4.9: 'versicolor',
                                        5.1: 'virginica'}},
                 1.6: 'versicolor',
                 1.8: 'virginica',
                 1.9: 'virginica',
                 2.0: 'virginica',
                 2.1: 'virginica',
                 2.2: 'virginica',
                 2.3: 'virginica',
                 2.4: 'virginica',
                 2.5: 'virginica'}}
```

In [103]:
```
def predict(inst,tree):
    for nodes in tree.keys():
        value = inst[nodes]
        tree = tree[nodes][value]
        prediction = 0
        if type(tree) is dict:
            prediction = predict(inst, tree)
        else:
            prediction = tree
            break
    return prediction
```

In [104]:
```
def test(data,tree):
    queries=data.to_dict('records')
    predicted=pd.DataFrame(columns=["predicted"])
    for i in range(len(data)):
        try:
          predicted.loc[i,"predicted"]=predict(queries[i],tree)
        except:
          predicted.loc[i,"predicted"]=df_train['species'].mode()[0]
    return predicted
```

In [105]:
```
y_pred=test(df_test.drop(columns=['species']),tree)
```

In [106]:
```
print("Accuracy:",metrics.accuracy_score(df_test['species'], y_pred)*100,"%")
```

```
Accuracy: 80.0 %
```

In [106]:
```

```