Project 2 Report

Jayam Sutariya

CS458

P2-1. Decision Tree

(a) Develop a decision tree based classifier to classify the 3 different types of Iris (Setosa, Versicolour, and Virginica).

In [34]:
```
# Codes for P2-1(a)
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold
import numpy as np

iris = datasets.load_iris()

X_train_set, X_holdout, y_train_set, y_holdout = train_test_split(iris.data, iris.target,
                                stratify = iris.target, random_state = 42, test_size = .25)

X = X_train_set
y = y_train_set

skf = StratifiedKFold(n_splits=5, shuffle = True)

scores = []
dt = DecisionTreeClassifier(criterion='gini', random_state = 42)

for train_index, test_index in skf.split(X, y):
    #print("Train index: {0}, \nTest index: {1}".format(train_index, test_index))
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    dt.fit(X_train, y_train)
    scores.append(dt.score(X_test, y_test))


print("The cross-validation scores using custom method are \n{0}".format(scores))
print("\nMean of k-fold scores is: {0}".format(np.mean(scores)))
```

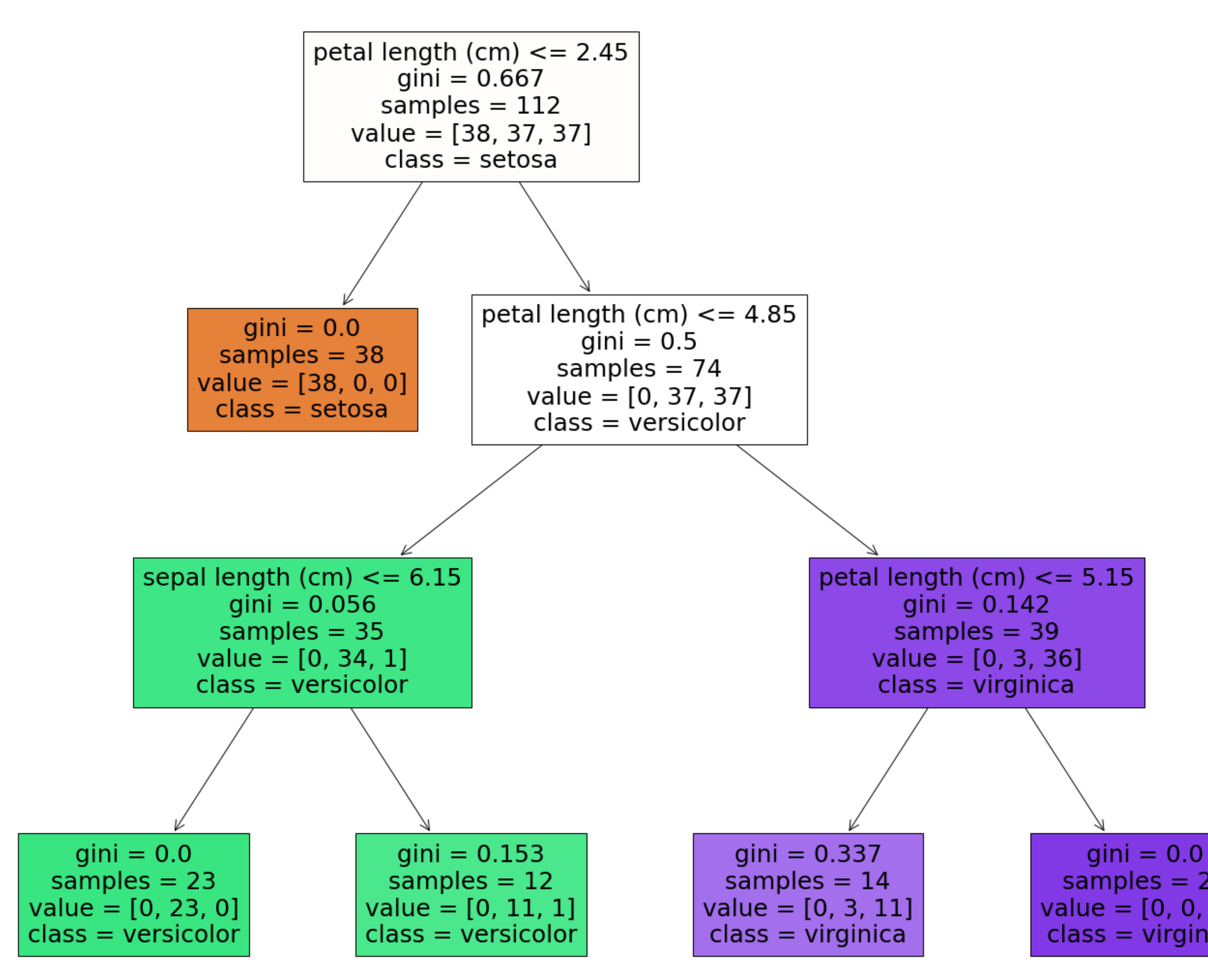
```
The cross-validation scores using custom method are
[0.9130434782608695, 0.8695652173913043, 0.9090909090909091, 0.9545454545454546, 0.9545454545454546]

Mean of k-fold scores is: 0.9201581027667984
```

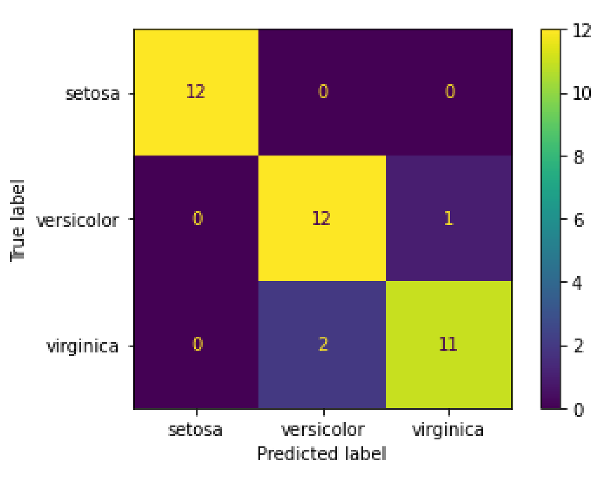
25% of the data was designated as the test set using the train_test_split() function. To make the training data balanced, StratifiedKFold with 5 folds and shuffled data was used to get the 5 folds. A for loop is traversed through the 5 folds, and the Decision Tree is generated with each fold of the training data.

(b) Optimize the parameters of your decision tree to maximize the classification accuracy. Show the confusion matrix of your decision tree. Plot your decision tree.

In [64]:
```
# Codes for P2-1(b)
from sklearn import datasets, tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, confusion_matrix

import matplotlib.pyplot as plt

import pydotplus
from IPython.display import Image

iris = datasets.load_iris()
class_names = iris.target_names

X_train_set, X_test, y_train_set, y_test = train_test_split(iris.data, iris.target,
                                stratify = iris.target, random_state = 42, test_size = .25)

X = X_train_set
y = y_train_set

dt = DecisionTreeClassifier(criterion='gini', random_state = 42, max_depth = 10, min_samples_leaf = 0.1)
dt.fit(X, y)

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(dt,
                   feature_names=iris.feature_names,
                   class_names=iris.target_names,
                   filled=True)

predY = dt.predict(X_test)
print('Accuracy on test data is %.2f' % (accuracy_score(y_test, predY)))
#'''
cm = confusion_matrix(y_test, predY, labels=[0, 1, 2])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot()
plt.show()
#'''
```

```
Accuracy on test data is 0.92
```



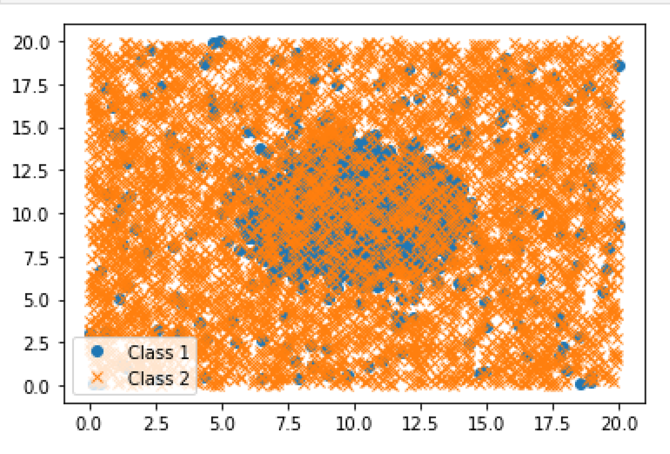


Not many adjustments were needed to be made to reach the average accuracy from the previous part. I adjusted the min_samples_leaf and the max_depth attribute until I reached the test data accuracy which was the closest tothe average accuracy from the last part.

P2-2. Model Overfitting

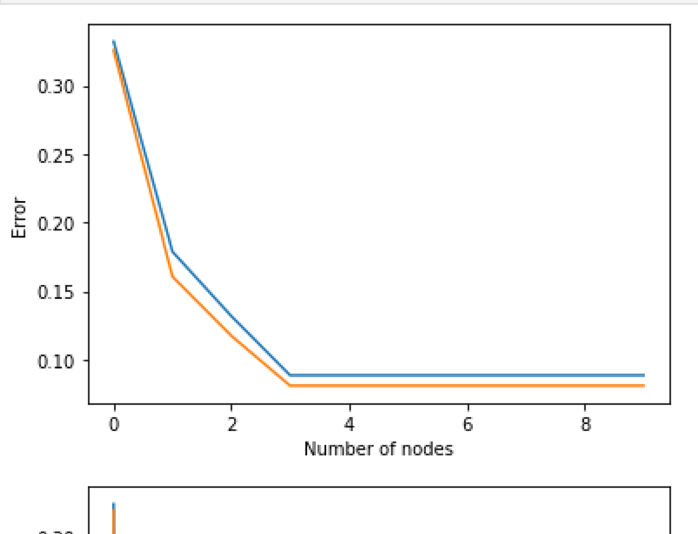
(a) Generate the dataset as in slide 56 in Chapter 3

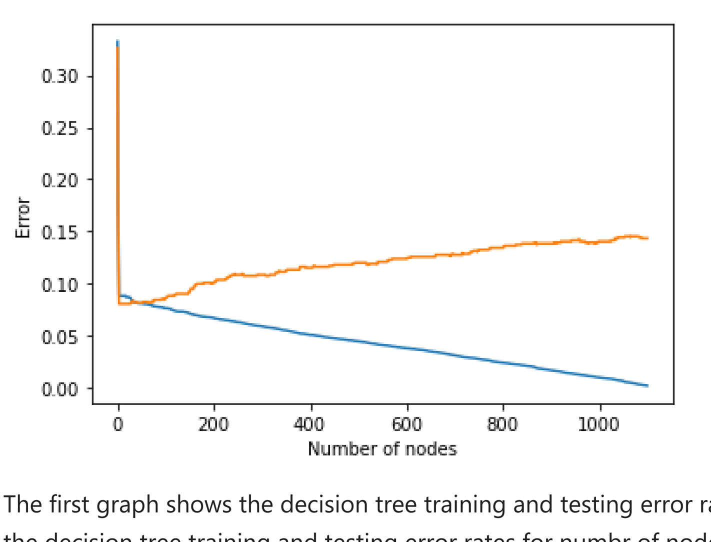
In [60]:
```
# Codes for P2-2(a)
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd

mean = [10.0, 10.0]
cov = [[2.0, 0], [0, 2.0]]

x1, y1 = np.random.multivariate_normal(mean, cov, 5000).T

xy_min = [0, 0]
xy_max = [20, 20]
ux1 = np.random.uniform(low = 0, high = 20, size = 200)
uy1 = np.random.uniform(low = 0, high = 20, size = 200)
class1x = np.concatenate((x1, ux1))
class1y = np.concatenate((y1, uy1))
class1l = np.full(5200, 1)

ux2 = np.random.uniform(low = 0, high = 20, size = 5200)
uy2 = np.random.uniform(low = 0, high = 20, size = 5200)
class2l = np.full(5200, 2)

column1 = np.concatenate((class1x, ux2))
column2 = np.concatenate((class1y, uy2))
label = np.concatenate((class1l, class2l))
#index = np.arange(0, 10400, 1)

#data = np.array((column1, column2, label))
df = pd.DataFrame()
df['x'] = pd.Series(column1)
df['y'] = pd.Series(column2)
df['class'] = pd.Series(label)

#data = [{'x': ux1, 'y': uy1, 'class': 1},
        #{'x': ux2, 'y': uy2, 'class': 2}]
#df = pd.DataFrame(data)

plt.plot(class1x, class1y, 'o', label = "Class 1")
plt.plot(ux2, uy2, 'x', label = "Class 2")
plt.legend()
plt.show()
```



(b) Randomly select 10% of the data as test dataset and the remaining 90% of the data as training dataset. Train decision trees by increasing the number of nodes of the decision trees until the training error becomes 0. Plot the training errors and the testing errors under different numbers of nodes and explain the model underfitting and model overfitting.

In [101…
```
# Codes for P2-2(b)
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X = df.drop(['class'], axis = 1).values
y = df['class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42, test_size = .1)

trainErr = []
testErr = []
for i in range(10):
    clf = DecisionTreeClassifier(criterion='entropy', random_state = 42, max_leaf_nodes = i+2)
    clf.fit(X_train, y_train)

    prediction1 = clf.predict(X_train)
    acc1 = accuracy_score(y_train, prediction1)
    trainErr.append(1 - acc1)

    prediction2 = clf.predict(X_test)
    acc2 = accuracy_score(y_test, prediction2)
    testErr.append(1 - acc2)

plt.plot(trainErr)
plt.plot(testErr)
plt.xlabel("Number of nodes")
plt.ylabel("Error")
plt.show()

trainErr2 = []
testErr2 = []
for i in range(1100):
    clf2 = DecisionTreeClassifier(criterion='entropy', random_state = 42, max_leaf_nodes = i+2)
    clf2.fit(X_train, y_train)

    prediction1 = clf2.predict(X_train)
    acc1 = accuracy_score(y_train, prediction1)
    trainErr2.append(1 - acc1)

    prediction2 = clf2.predict(X_test)
    acc2 = accuracy_score(y_test, prediction2)
    testErr2.append(1 - acc2)

plt.plot(trainErr2)
plt.plot(testErr2)
plt.xlabel("Number of nodes")
plt.ylabel("Error")
plt.show()
```





The first graph shows the decision tree training and testing error rates for number of nodes ranging from 0 to 10. The second graph shows the decision tree training and testing error rates for numbr of nodes ranging from 0 to 1100.

As you can see in the first graph, the training and testing errors are very similar and consisten all the way. The only problem there is that, when the number of nodes are less than around 5, the errors are extremely high. Meaning that the model is underfitting as the error is extremely high.

On the other hand, in the second graph, the training and testing errors are very similar and consistent upto a certain point. However, after a certain number of nodes, the testing error seems to increase, whereas, the training error continues to decrease until it reaches 0. This is because the model is overfitting after a certain number of nodes. To optimize the decision tree classification accuracy, and to avoid overfitting and underfitting the model, the number of nodes must be kept in check.

P2-3. Text Documents Classification

(a) Load the following 4 categories from the 20 newsgroups dataset: categories = ['rec.autos', 'talk.religion.misc', 'comp.graphics', 'sci.space']. Print the number of documents in the training dataset and the test dataset. Print the number of attributes in the training dataset.

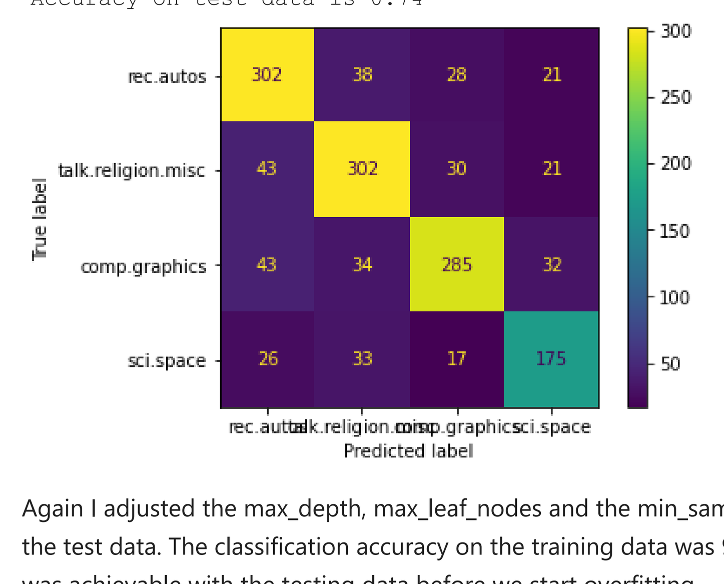
In [28]:
```
# Codes for P2-3(a)
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd
import numpy as np

categories = ['rec.autos', 'talk.religion.misc', 'comp.graphics', 'sci.space']
trainData = fetch_20newsgroups(subset = 'train', categories=categories)
testData = fetch_20newsgroups(subset = 'test', categories=categories)

print("The number of documents in the training data:", len(trainData.target))
print("The number of documents in the testing data:", len(testData.target))
#print(trainData.DESCR)

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(trainData.data)
print("The number of attributes in the training data:", vectors.shape[1])
```

```
The number of documents in the training data: 2148
The number of documents in the testing data: 1430
The number of attributes in the training data: 34948
```

(b) Optimize the parameters of your decision tree to maximize the classification accuracy. Show the confusion matrix of your decision tree.

In [113…
```
# Codes for P2-3(b)
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, confusion_matrix
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

categories = ['rec.autos', 'talk.religion.misc', 'comp.graphics', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset = 'train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset = 'test', categories=categories)

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(newsgroups_train.data)
vectors_test = vectorizer.transform(newsgroups_test.data)

x1 = vectors
y1 = newsgroups_train.target
x2 = vectors_test
y2 = newsgroups_test.target

dt = DecisionTreeClassifier(criterion='entropy', random_state = 42, max_leaf_nodes = 150)
dt.fit(x1, y1)

predY = dt.predict(x2)
print('Accuracy on test data is %.2f' % (accuracy_score(y2, predY)))

cm = confusion_matrix(y2, predY, labels=[0, 1, 2, 3])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=categories)
disp.plot()
plt.show()
```

```
Accuracy on test data is 0.74
```



Again I adjusted the max_depth, max_leaf_nodes and the min_samples_leaf attributes to try to achieve a higher classification accuracy on the test data. The classification accuracy on the training data was 97%. However, the accuracy printed above is the highest accuracy that was achievable with the testing data before we start overfitting.