da5

June 4, 2020

DIGITAL ASSIGNMENT 5

Name : Sidhnatha Poddar

Reg 17BCE2044

0.0.1 Content:

- KNN
- Naive Bayes
- Descision Tree

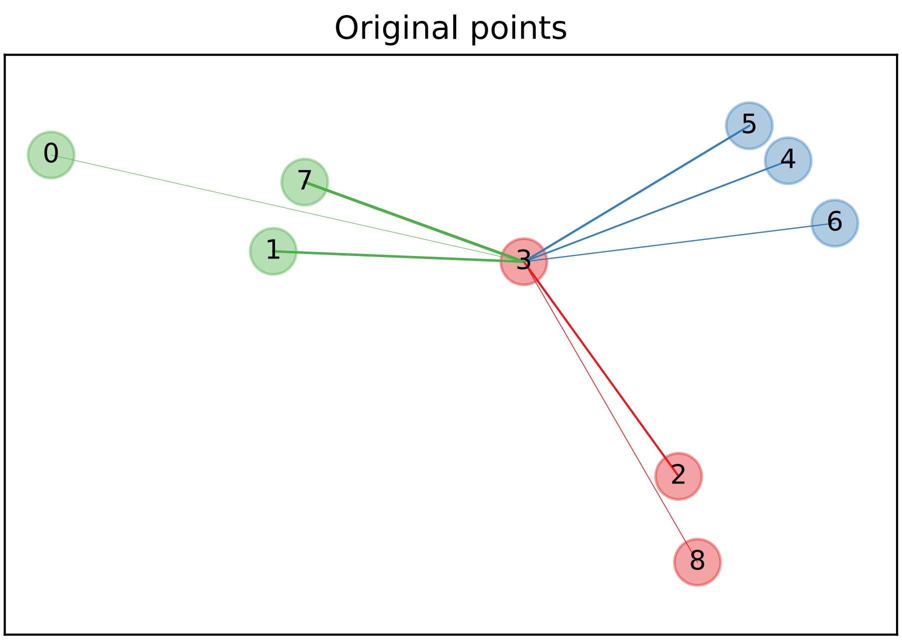
1 KNN

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
#from sklearn.neighbors import NeighborhoodComponentsAnalysis
from matplotlib import cm
from sklearn.utils.fixes import logsumexp
import sklearn.neighbors
print(__doc__)
import pandas as pd
from numpy import hstack,dstack,vstack
```

```
Automatically created module for IPython interactive environment
```

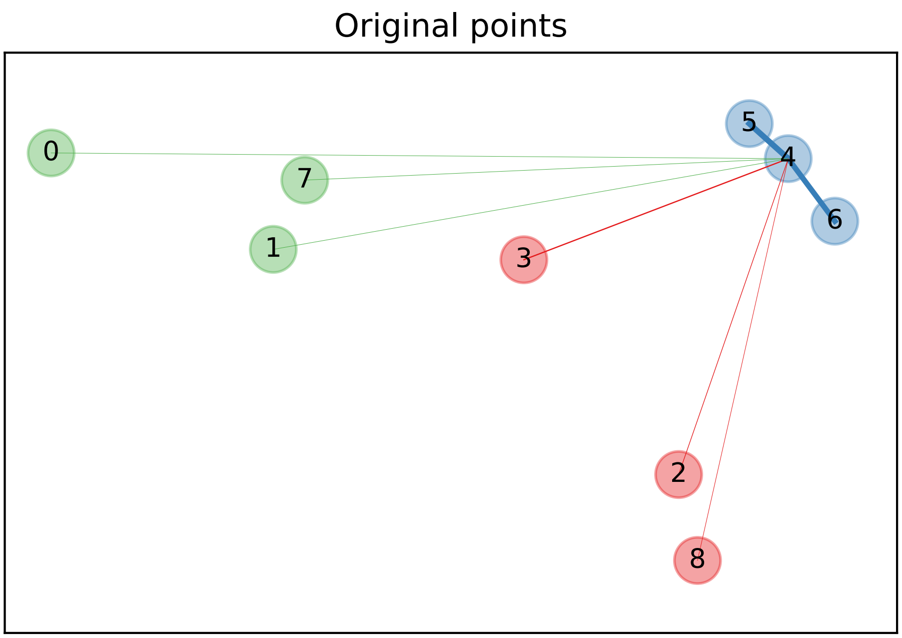
[8]:
```
X, y = make_classification(n_samples=9, n_features=2,
    n_informative=2,n_redundant=0,
    n_classes=3, n_clusters_per_class=1,class_sep=1.0, random_state=0)
plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)
ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.axis('equal') # so that boundaries are displayed correctly as circles
```

```
def link_thickness_i(X, i):
    diff_embedded = X[i] - X
    dist_embedded = np.einsum('ij,ij->i', diff_embedded,
    diff_embedded)
    dist_embedded[i] = np.inf
    # compute exponentiated distances (use the log-sum-exp trick to
    # avoid numerical instabilities
    exp_dist_embedded = np.exp(-dist_embedded -
    logsumexp(-dist_embedded))
    return exp_dist_embedded

def relate_point(X, i, ax):
    pt_i = X[i]
    for j, pt_j in enumerate(X):
        thickness = link_thickness_i(X, i)
        if i != j:
            line = ([pt_i[0], pt_j[0]], [pt_i[1], pt_j[1]])
            ax.plot(*line, c=cm.Set1(y[j]),
            linewidth=5*thickness[j])
i = 3
relate_point(X, i, ax)
plt.show()
```



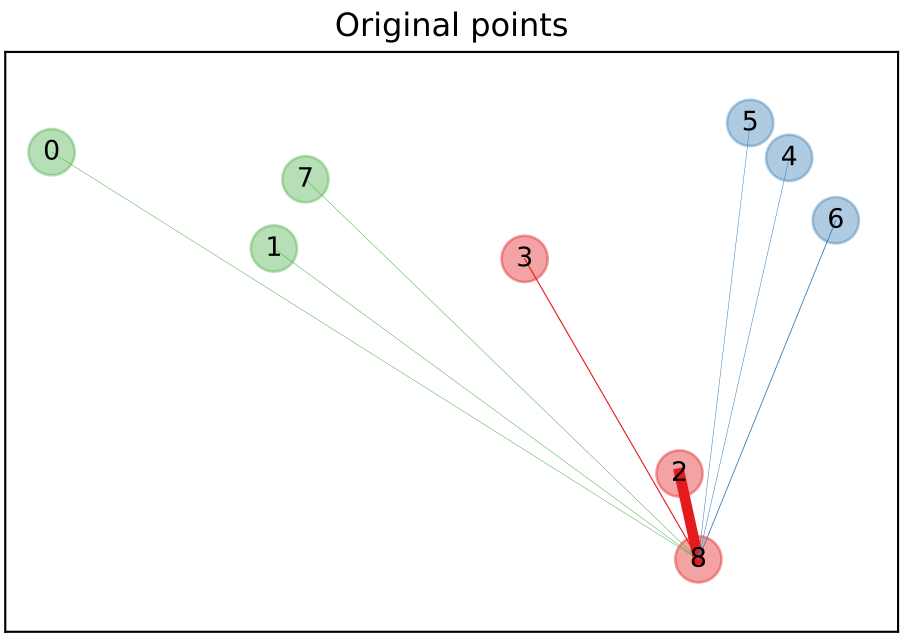
[9]:
```
X, y = make_classification(n_samples=9, n_features=2, n_informative=2,
    n_redundant=0, n_classes=3, n_clusters_per_class=1,
```

```
    class_sep=1.0, random_state=0)
plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)
ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.axis('equal') # so that boundaries are displayed correctly as circles
i = 4
relate_point(X, i, ax)
plt.show()
```



[10]:
```
X, y = make_classification(n_samples=9, n_features=2, n_informative=2,
    n_redundant=0, n_classes=3, n_clusters_per_class=1,
    class_sep=1.0, random_state=0)
plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)
ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
```

```
ax.axis('equal') # so that boundaries are displayed correctly as circles
i = 8
relate_point(X, i, ax)
plt.show()
```



[13]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

[14]:
```
n_neighbors = 1
dataset = datasets.load_iris()
X, y = dataset.data, dataset.target
# we only take two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = X[:, [0, 2]]
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪test_size=0.7)
pd.DataFrame(X_train).head()
```

[14]:

	0	1
0	6.4	5.6
1	5.4	1.5

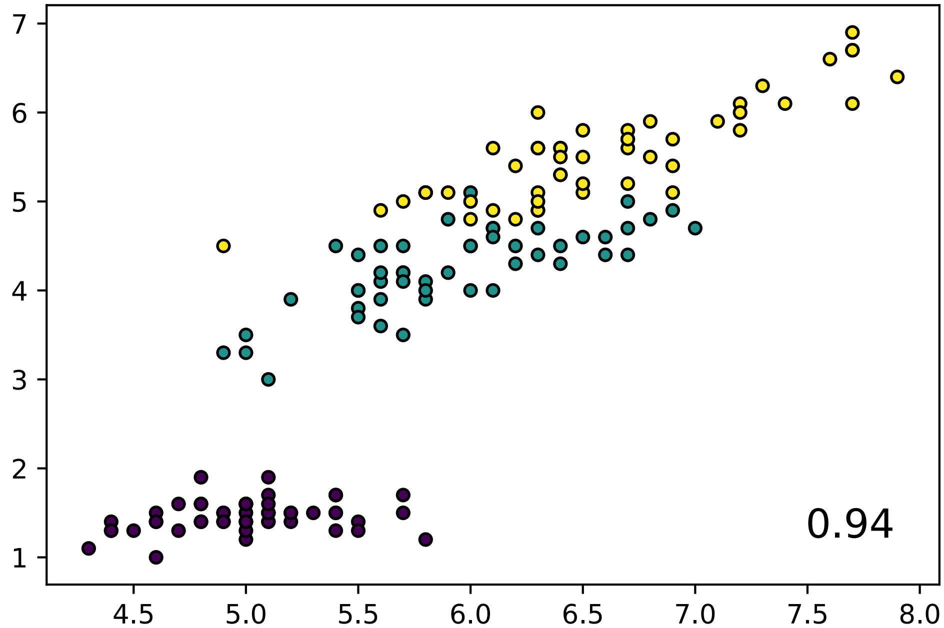
```
2  6.3  4.9
3  6.3  4.7
4  7.4  6.1
```

[22]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
metric_params=None, n_jobs=1, n_neighbors=5, p=2,weights='uniform')
print("Predictions form the classifier:")
print(knn.predict(X_test))
print("Target values:")
print(y_test)
knn.score(X_test,y_test)
```

```
Predictions form the classifier:
[1 1 1 2 1 0 0 2 1 0 0 2 2 1 0 2 1 2 1 1 2 1 0 0 2 2 0 1 1 1 1 2 0 2 0 0 1
 0 1 2 1 0 0 0 0 1 2 1 1 1 1 1 1 0 0 1 1 2 1 0 1 0 0 2 0 2 2 1 1 0 0 2 0 1 1
 0 0 1 0 0 2 2 2 0 1 2 0 2 2 2 0 0 2 0 2 1 2 2 1 1 1 1 2 2 0 1]
Target values:
[1 1 1 2 1 0 0 2 1 0 0 2 2 1 0 2 1 2 1 1 2 1 0 0 1 2 0 2 1 1 1 2 0 2 0 0 1
 0 1 2 1 0 0 0 0 1 2 1 1 1 2 1 0 0 1 1 2 1 0 1 0 0 2 0 2 2 2 2 0 0 2 0 1 1
 0 0 1 0 0 2 2 2 0 1 2 0 2 2 2 0 0 2 0 2 1 2 2 2 1 1 1 2 2 0 1]
```

[22]: 0.9428571428571428

[23]:
```
clf=knn
if True:
 clf.fit(X_train, y_train)
 score = clf.score(X_test, y_test)
 plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', s=20)
 plt.text(0.9, 0.1, '{:.2f}'.format(score), size=15,
 ha='center', va='center', transform=plt.gca().transAxes)
plt.show()
```



[25]:

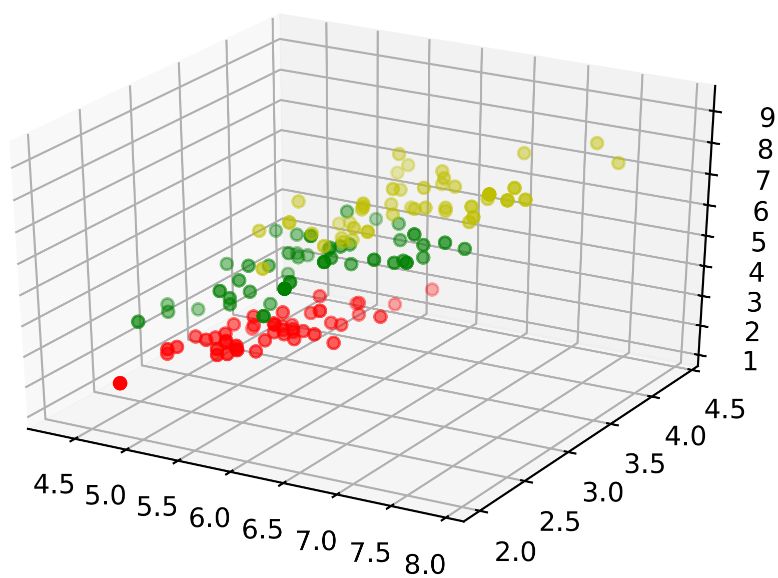
```
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target
print(iris_data[0], iris_data[79], iris_data[100])
print(iris_labels[0], iris_labels[79], iris_labels[100])
np.random.seed(42)
indices = np.random.permutation(len(iris_data))
n_training_samples = 12
learnset_data = iris_data[indices[:-n_training_samples]]
learnset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = iris_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]
print(learnset_data[:4], learnset_labels[:4])
print(testset_data[:4], testset_labels[:4])
```

```
[5.1 3.5 1.4 0.2] [5.7 2.6 3.5 1. ] [6.3 3.3 6.  2.5]
0 1 2
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]] [1 0 2 1]
[[5.7 2.8 4.1 1.3]
 [6.5 3.  5.5 1.8]
 [6.3 2.3 4.4 1.3]
```

```
[6.4 2.9 4.3 1.3]] [1 2 1 1]
```

[30]:

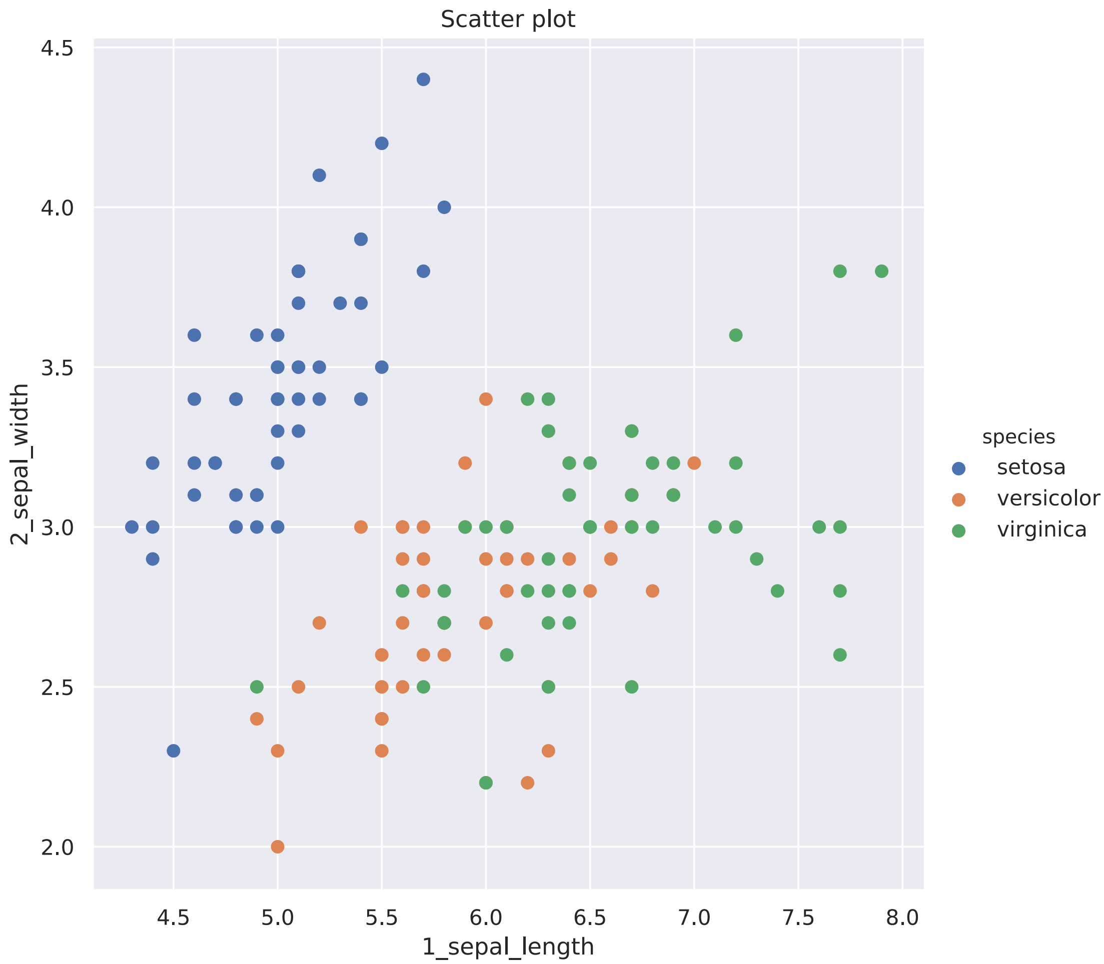
```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
colours = ("r", "b")
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(learnset_data)):
        if learnset_labels[i] == iclass:
            X[iclass][0].append(learnset_data[i][0])
            X[iclass][1].append(learnset_data[i][1])
            X[iclass][2].append(sum(learnset_data[i][2:]))
colours = ("r", "g", "y")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for iclass in range(3):
 ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])
plt.show()
```



2 Gaussian Naive Bayes Classifier: Iris data set

[32]:
```
import pandas as pd
from matplotlib import pyplot as plt
import matplotlib.colors as colors
import seaborn as sns
import itertools
from scipy.stats import norm
import scipy.stats
from sklearn.naive_bayes import GaussianNB

%matplotlib inline
sns.set()
```

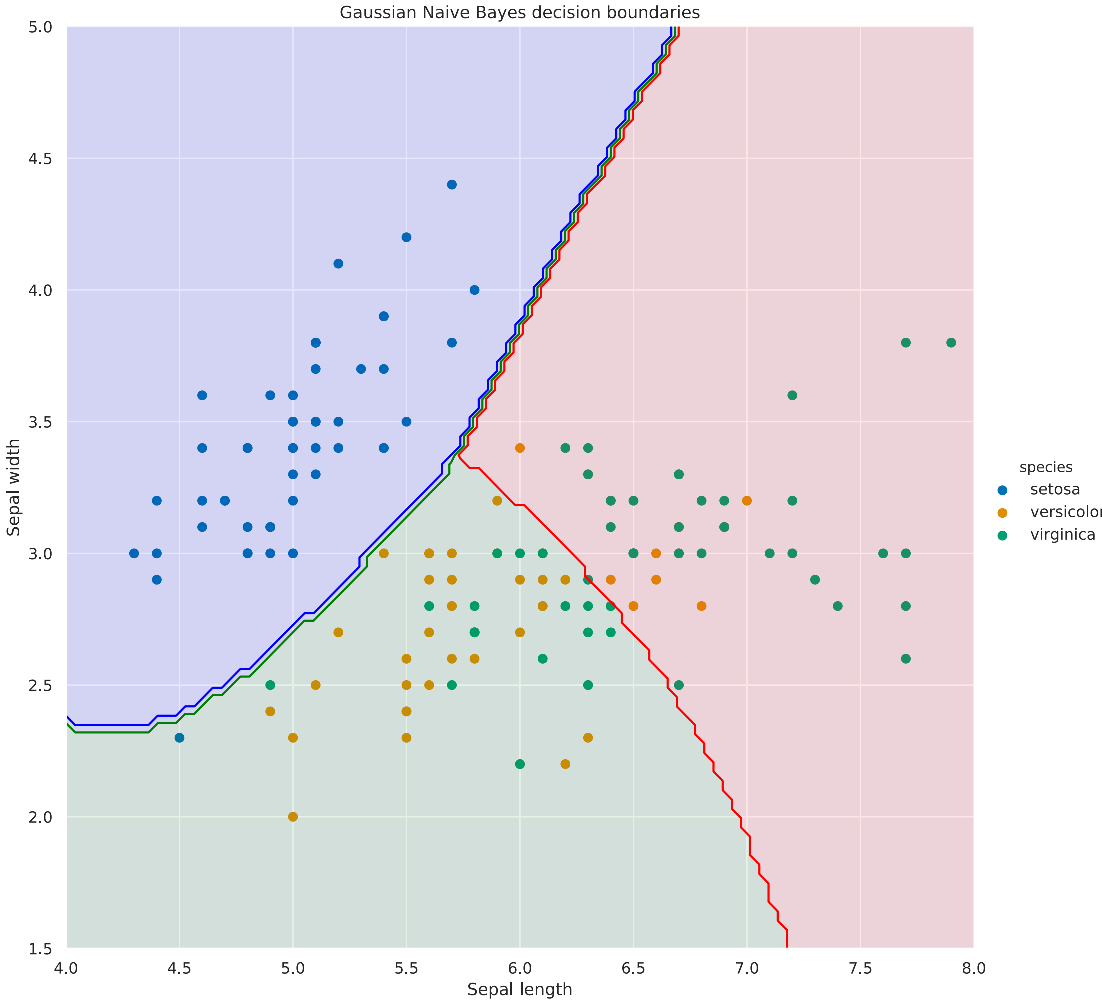
[33]:
```
#Load the data set
iris = sns.load_dataset("iris")
iris = iris.rename(index = str, columns = {'sepal_length':
↪'1_sepal_length','sepal_width':'2_sepal_width', 'petal_length':
↪'3_petal_length', 'petal_width':'4_petal_width'})

#Plot the scatter of sepal length vs sepal width
sns.FacetGrid(iris, hue="species", size=7) .map(plt.scatter,"1_sepal_length",␣
↪"2_sepal_width", )  .add_legend()
plt.title('Scatter plot')
df1 = iris[["1_sepal_length", "2_sepal_width",'species']]
```



[36]:

```
from sklearn.naive_bayes import GaussianNB

#Setup X and y data
X_data = df1.iloc[:,0:2]
y_labels = df1.iloc[:,2].replace({'setosa':0,'versicolor':1,'virginica':2}).
 ↪copy()

#Fit model
model_sk = GaussianNB(priors = None)
model_sk.fit(X_data,y_labels)


# Our 2-dimensional classifier will be over variables X and Y
N = 100
X = np.linspace(4, 8, N)
Y = np.linspace(1.5, 5, N)
X, Y = np.meshgrid(X, Y)
```

```
#fig = plt.figure(figsize = (10,10))
#ax = fig.gca()
color_list = ['Blues','Greens','Reds']
my_norm = colors.Normalize(vmin=-1.,vmax=1.)

g = sns.FacetGrid(iris, hue="species", size=10, palette = 'colorblind') .
↪map(plt.scatter, "1_sepal_length", "2_sepal_width",)  .add_legend()
my_ax = g.ax


#Computing the predicted class function for each value on the grid
zz = np.array(  [model_sk.predict( [[xx,yy]])[0] for xx, yy in zip(np.ravel(X),␣
↪np.ravel(Y)) ] )


#Reshaping the predicted class into the meshgrid shape
Z = zz.reshape(X.shape)


#Plot the filled and boundary contours
my_ax.contourf( X, Y, Z, 2, alpha = .1, colors = ('blue','green','red'))
my_ax.contour( X, Y, Z, 2, alpha = 1, colors = ('blue','green','red'))

# Addd axis and title
my_ax.set_xlabel('Sepal length')
my_ax.set_ylabel('Sepal width')
my_ax.set_title('Gaussian Naive Bayes decision boundaries')

plt.show()
```

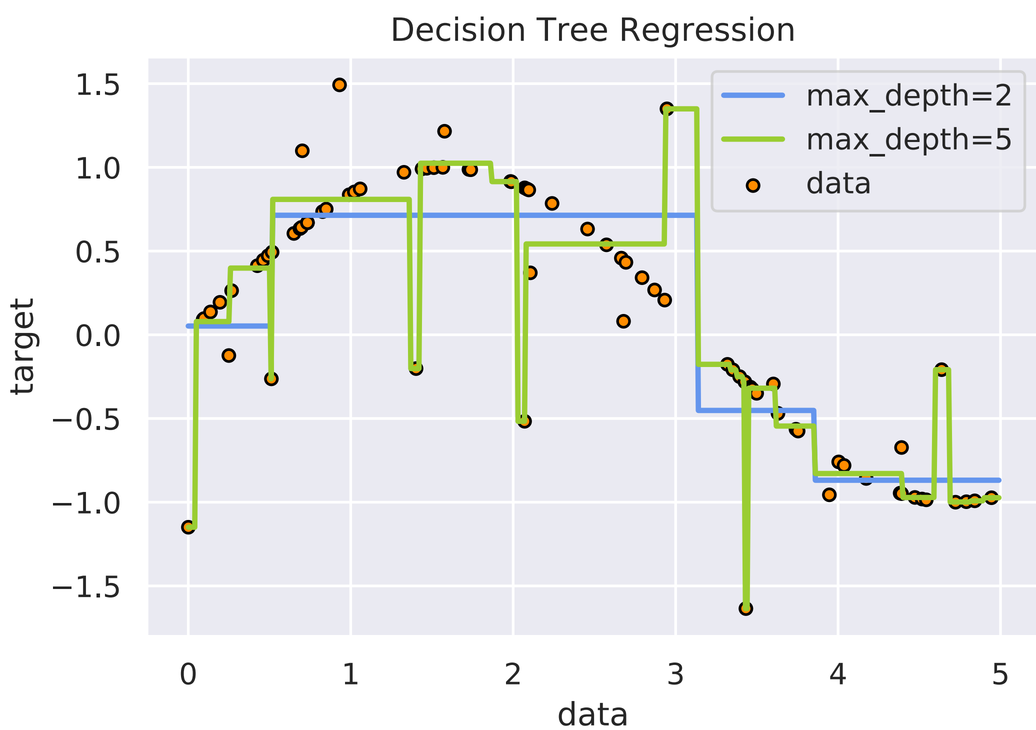


3 descision tree classification using random generated dataset

[37]:
```
print(__doc__)
# Import the necessary modules and libraries
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))
# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=5)
```

```
regr_1.fit(X, y)
regr_2.fit(X, y)
# Predict
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
# Plot the results
plt.figure()
plt.scatter(X, y, s=20, edgecolor="black",
c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue",
label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

```
Automatically created module for IPython interactive environment
```



4 Descision tree Classification using Dibabetes Dataset

[47]:
```
import pandas as pd
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sn
import matplotlib.pyplot as plt
```

[39]:
```
data=pd.read_csv('d.csv')
data.head()
```

[39]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[40]:
```
data.info()
```

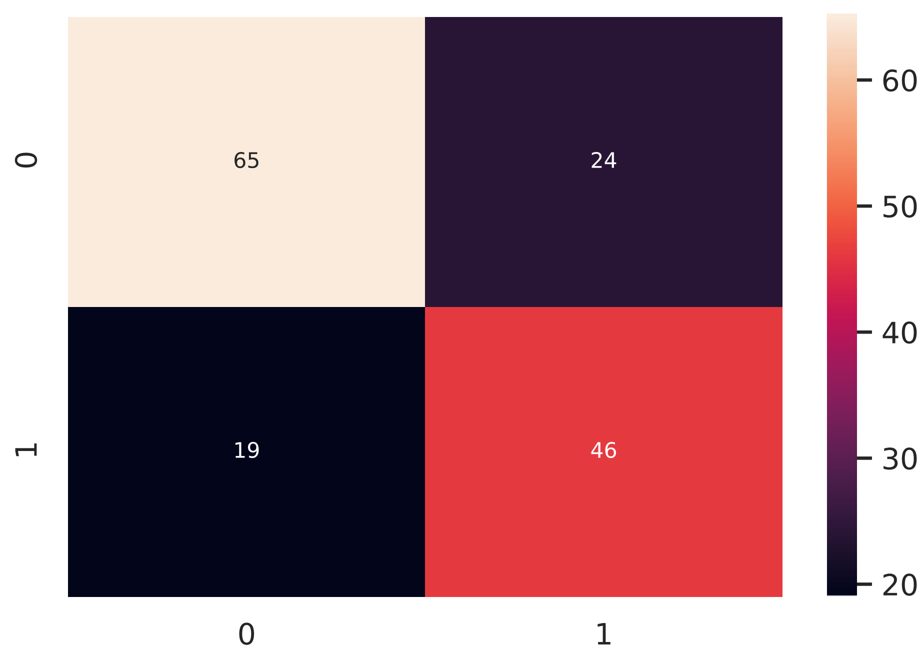
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[44]:
```
X=data.iloc[:,:8]
y=data.iloc[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)
```

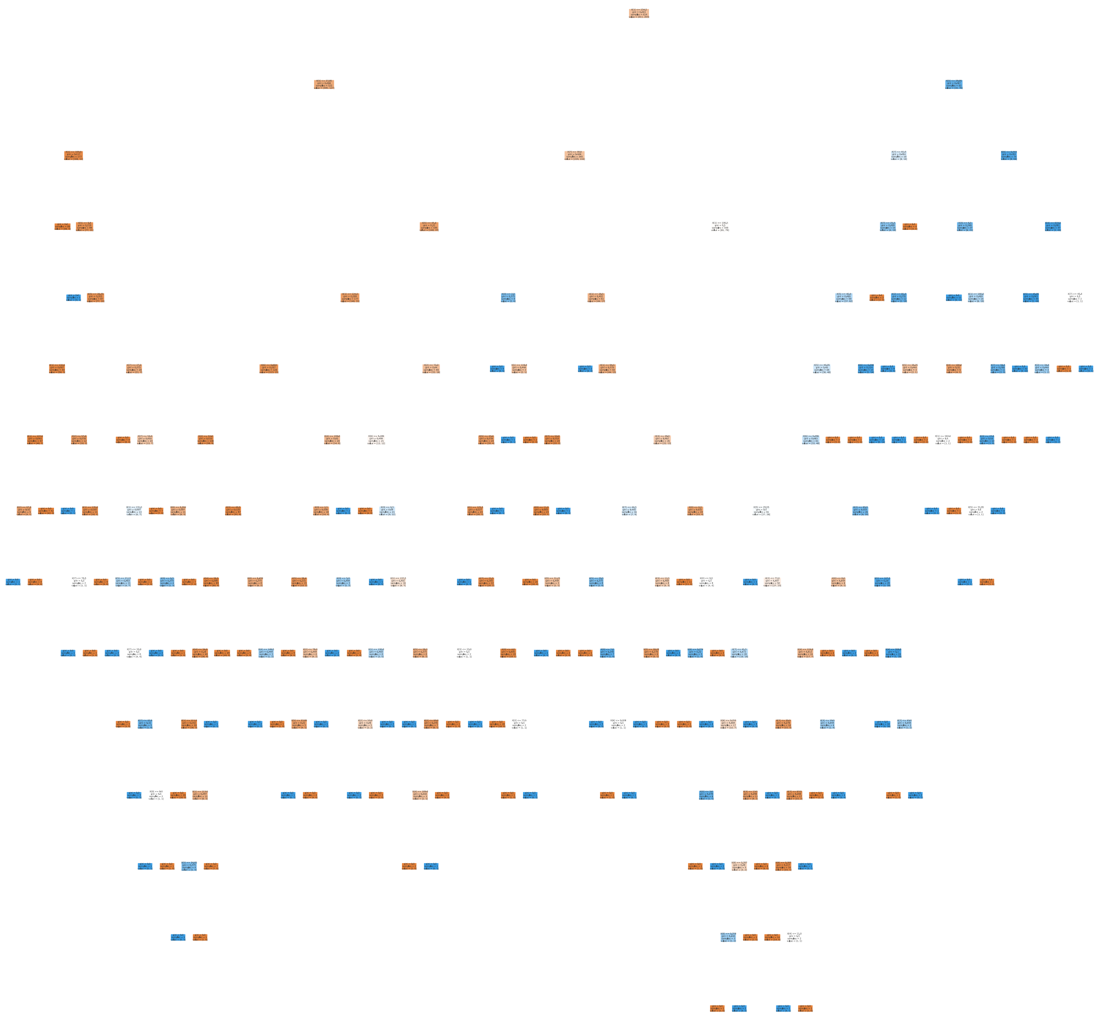
[48]:
```
clf = tree.DecisionTreeClassifier()# defining classifier
clf = clf.fit(X_train, y_train) #fitting model
y_pred = clf.predict(X_test)
```

[50]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
sn.heatmap(confusion_matrix(y_test, y_pred), annot=True, annot_kws={"size": 8})
plt.show()
```

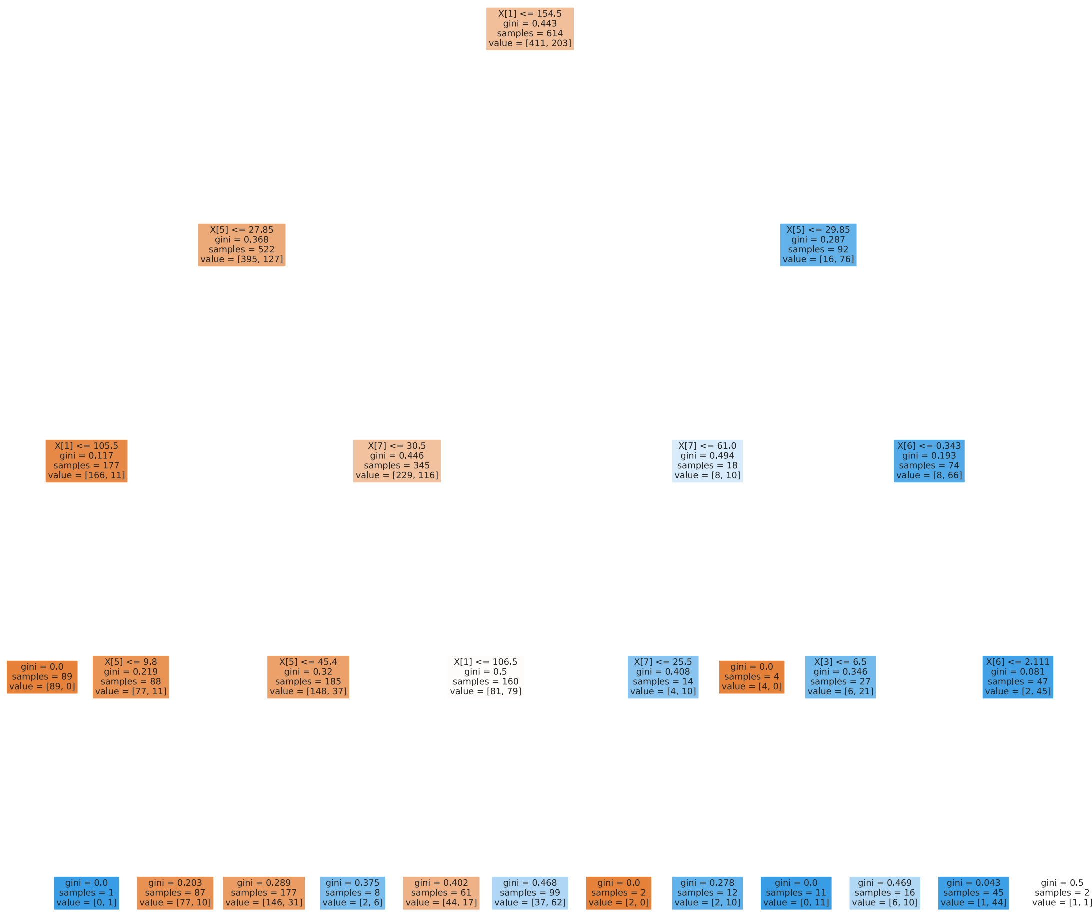
```
[[65 24]
 [19 46]]
              precision    recall  f1-score   support

           0       0.77      0.73      0.75        89
           1       0.66      0.71      0.68        65

    accuracy                           0.72       154
   macro avg       0.72      0.72      0.72       154
weighted avg       0.72      0.72      0.72       154
```



[52]:
```
plt.figure(figsize=(10,10))
tree.plot_tree(clf, filled=True)
plt.show()
```



[54]:
```
clf = tree.DecisionTreeClassifier(max_depth=4)# defining classifier
clf = clf.fit(X_train, y_train) #fitting model
y_pred = clf.predict(X_test)
plt.figure(figsize=(20,20))
tree.plot_tree(clf, filled=True)
plt.show()
```


X[1] <= 154.5
gini = 0.443
samples = 614
value = [411, 203]
X[5] <= 27.85
gini = 0.368
samples = 522
value = [395, 127]
X[5] <= 29.85
gini = 0.287
samples = 92
value = [16, 76]
X[1] <= 105.5
gini = 0.117
samples = 177
value = [166, 11]
X[7] <= 30.5
gini = 0.446
samples = 345
value = [229, 116]
X[7] <= 61.0
gini = 0.494
samples = 18
value = [8, 10]
X[6] <= 0.343
gini = 0.193
samples = 74
value = [8, 66]
gini = 0.0
samples = 89
value = [89, 0]
X[5] <= 9.8
gini = 0.219
samples = 88
value = [77, 11]
X[5] <= 45.4
gini = 0.32
samples = 185
value = [148, 37]
X[1] <= 106.5
gini = 0.5
samples = 160
value = [81, 79]
X[7] <= 25.5
gini = 0.408
samples = 14
value = [4, 10]
gini = 0.0
samples = 4
value = [4, 0]
X[3] <= 6.5
gini = 0.346
samples = 27
value = [6, 21]
X[6] <= 2.111
gini = 0.081
samples = 47
value = [2, 45]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.203
samples = 87
value = [77, 10]
gini = 0.289
samples = 177
value = [146, 31]
gini = 0.375
samples = 8
value = [2, 6]
gini = 0.402
samples = 61
value = [44, 17]
gini = 0.468
samples = 99
value = [37, 62]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.278
samples = 12
value = [2, 10]
gini = 0.0
samples = 11
value = [0, 11]
gini = 0.469
samples = 16
value = [6, 10]
gini = 0.043
samples = 45
value = [1, 44]
gini = 0.5
samples = 2
value = [1, 1]

[ ]: