Final Artificial intelligent

NAME: MD. NAYEEM MOLLA

ID:20183290381

MAJOR: COMPUTER SCIENCE &TECHNOLOGY

SCHOOL: School of Information Science & Engineering

Subject: Artificial intelligent

Date: 6/5/2022

Final Project

Introduction

This is my final project for the Artificial intelligent course. I did complete all homework and attend each and every class it was really a wonderful learning course for me because I want to do my masters also in Artificial intelligence. In this project, I will do classification for the iris dataset from chapter 1 then I will use SVM and random forest respectively, and last, I will use matrices to check the accuracy to evaluate my model.

Use the Iris dataset in Chapter 1 (from sklearn. datasets import load_iris) to do the classification

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor). These measures were used to create a linear discriminant model to classify the species. Classification is the process of categorizing a set of data into classes. It can be done on both structured and unstructured data. Predicting the class of provided data points is the first step in the procedure. The classes are also known as the target, label, or categories. Approximating the mapping function from discrete input variables to discrete output variables is the problem of classification predictive modeling. The basic purpose is to determine which class/category the new data belongs to. Now I will classify iris dataset:

Name: MD NAYEEM MOLLA

ID;20183290381

[1]

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from pandas.plotting import parallel_coordinates
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
```

[2]

```
# load through url
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
dataset = pd.read_csv(url, names = attributes)
dataset.columns = attributes
```

[2]

```
# load through url
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
attributes = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]
dataset = pd.read_csv(url, names = attributes)
dataset.columns = attributes
```


```
dataset
```

0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```python
data.head(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[6]
```python
data.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

dtype: object

[7]
```python
# number of instances in each class
data.groupby('species').size()
```

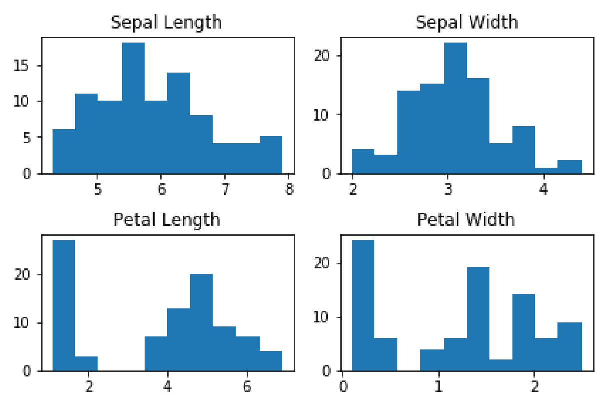
```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

[8]
```python
# Take out a test set
train, test = train_test_split(data, test_size = 0.4, stratify = data['species'], random_state = 42)
```

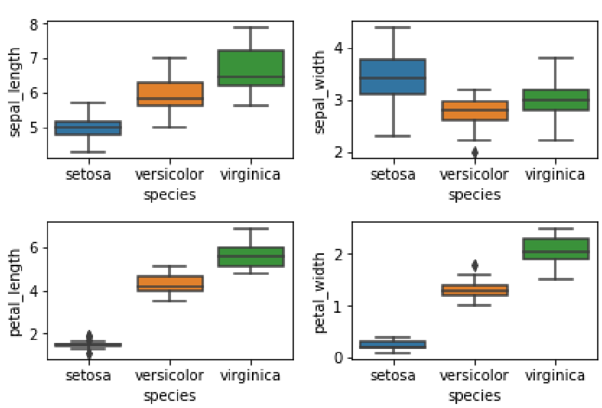
[9]
```python
# number of instances in each class in training data
train.groupby('species').size()
```

```
species
setosa        30
versicolor    30
virginica     30
dtype: int64
```

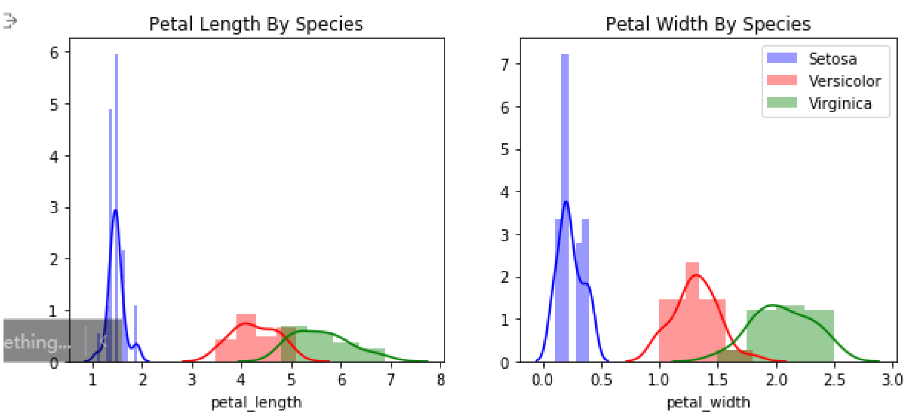
```python
# histograms
n_bins = 10
fig, axs = plt.subplots(2, 2)
axs[0,0].hist(train['sepal_length'], bins = n_bins);
axs[0,0].set_title('Sepal Length');
axs[0,1].hist(train['sepal_width'], bins = n_bins);
axs[0,1].set_title('Sepal Width');
axs[1,0].hist(train['petal_length'], bins = n_bins);
axs[1,0].set_title('Petal Length');
axs[1,1].hist(train['petal_width'], bins = n_bins);
axs[1,1].set_title('Petal Width');

# add some spacing between subplots
fig.tight_layout(pad=1.0);
```



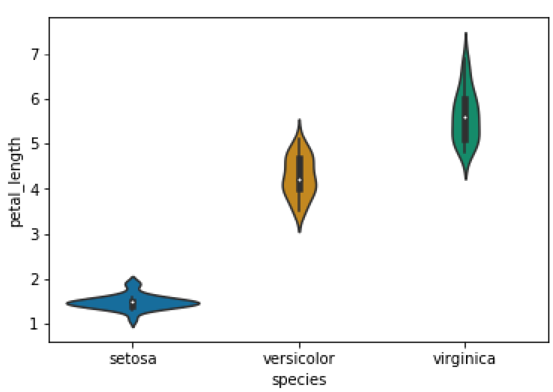

```python
# boxplots using seaborn
fig, axs = plt.subplots(2, 2)
fn = ["sepal_length", "sepal_width", "petal_length", "petal_width"]
cn = ['setosa', 'versicolor', 'virginica']
sns.boxplot(x = 'species', y = 'sepal_length', data = train, order = cn, ax = axs[0,0]);
sns.boxplot(x = 'species', y = 'sepal_width', data = train, order = cn, ax = axs[0,1]);
sns.boxplot(x = 'species', y = 'petal_length', data = train, order = cn, ax = axs[1,0]);
sns.boxplot(x = 'species', y = 'petal_width', data = train,  order = cn, ax = axs[1,1]);
# add some spacing between subplots
fig.tight_layout(pad=1.0);
```



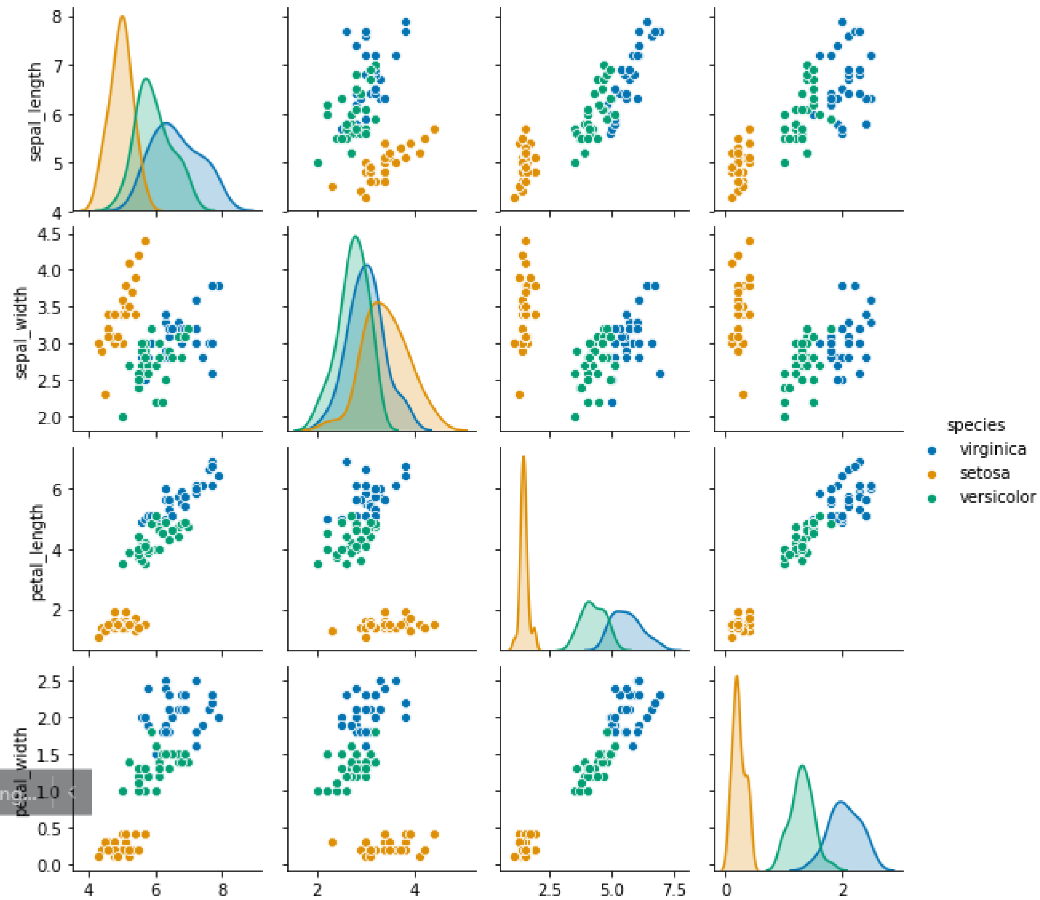
```python

# right off the bat, we see that petal length/width can separate setosa from the others
# histogram by species
setosa_pl = train.loc[train.species=='setosa', 'petal_length']
versicolor_pl = train.loc[train.species=='versicolor', 'petal_length']
virginica_pl = train.loc[train.species=='virginica', 'petal_length']
setosa_pw = train.loc[train.species=='setosa', 'petal_width']
versicolor_pw = train.loc[train.species=='versicolor', 'petal_width']
virginica_pw = train.loc[train.species=='virginica', 'petal_width']

fig, axs = plt.subplots(1, 2)
# set figure size
fig.set_size_inches(10,4)
ax1 = sns.distplot(setosa_pl, color="blue", label="Setosa", ax = axs[0]);
ax1.set_title('Petal Length By Species')
ax1 = sns.distplot(versicolor_pl, color="red", label="Versicolor", ax = axs[0]);
ax1 = sns.distplot(virginica_pl, color="green", label="Virginica", ax = axs[0]);

ax2 = sns.distplot(setosa_pw, color="blue", label="Setosa", ax = axs[1]);
ax2.set_title('Petal Width By Species')
ax2 = sns.distplot(versicolor_pw, color="red", label="Versicolor", ax = axs[1]);
ax2 = sns.distplot(virginica_pw, color="green", label="Virginica", ax = axs[1]);

plt.legend();
```



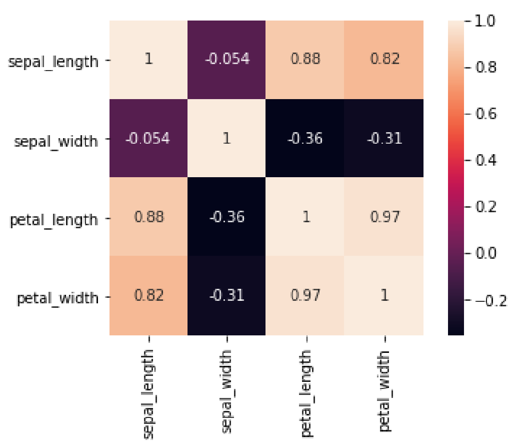

```python
sns.violinplot(x="species", y="petal_length", data=train, size=5, order = cn, palette = 'colorbli
```



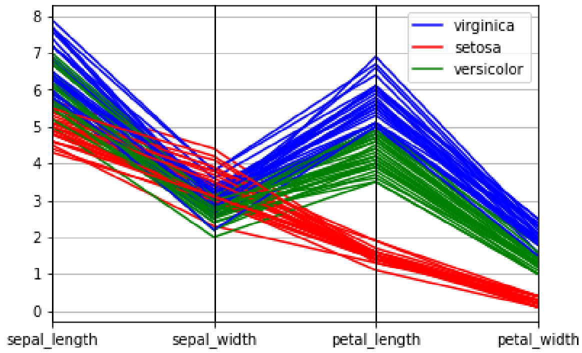
```python
# bivariate relationship
# scatterplot matrix
sns.pairplot(train, hue="species", height = 2, palette = 'colorblind');
```




```python
# correlation matrix
corrmat = train.corr()
sns.heatmap(corrmat, annot = True, square = True);
```



```python
# parallel coordinates
parallel_coordinates(train, "species", color = ['blue', 'red', 'green']);
```




```python
# first try decision tree
mod_dt = DecisionTreeClassifier(max_depth = 3, random_state = 1)
mod_dt.fit(X_train,y_train)
prediction=mod_dt.predict(X_test)
print('The accuracy of the Decision Tree is',"{:.3f}".format(metrics.accuracy_score(prediction,y_
```

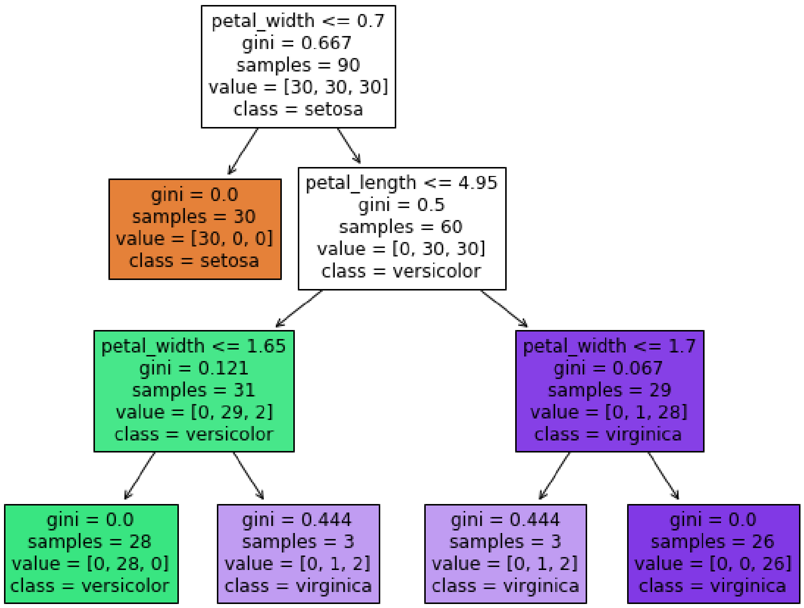
```
The accuracy of the Decision Tree is 0.983
```

[19]

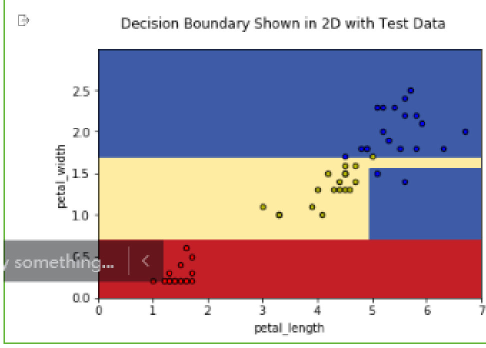
```python
mod_dt.feature_importances_
```

```
array([0.        , 0.        , 0.42430866, 0.57569134])
```

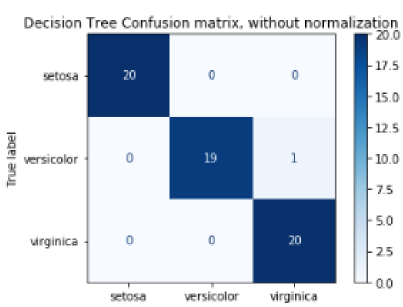
```
# set figure size
plt.figure(figsize = (10,8))
plot_tree(mod_dt, feature_names = fn, class_names = cn, filled = True);
```



```python
# plot decision boundary for pedal width vs pedal length
plot_step = 0.01
plot_colors = "ryb"
xx, yy = np.meshgrid(np.arange(0, 7, plot_step), np.arange(0, 3, plot_step))
plt.tight_layout(h_pad=1, w_pad=1, pad=2.5)

selected_predictors = ["petal_length", "petal_width"]
mod_dt_1 = DecisionTreeClassifier(max_depth = 3, random_state = 1)
y_train_en = y_train.replace({'setosa':0,'versicolor':1,'virginica':2}).copy()
mod_dt_1.fit(X_train[selected_predictors],y_train_en)

pred_all = mod_dt_1.predict(np.c_[xx.ravel(), yy.ravel()])
pred_all = pred_all.reshape(xx.shape)

graph = plt.contourf(xx, yy, pred_all, cmap=plt.cm.RdYlBu)

plt.xlabel(selected_predictors[0])
plt.ylabel(selected_predictors[1])

# plot test data points
n_class = 3
for i, color in zip(cn, plot_colors):
    temp = np.where(y_test == i)
    idx = [elem for elems in temp for elem in elems]
    plt.scatter(X_test.iloc[idx, 2], X_test.iloc[idx, 3], c=color,
                label=y_test, cmap=plt.cm.RdYlBu, edgecolor='black', s=20)

plt.suptitle("Decision Boundary Shown in 2D with Test Data")
plt.axis("tight");
```




```python
# confusion matrix
# one versicolor misclassified
disp = metrics.plot_confusion_matrix(mod_dt, X_test, y_test,
                                     display_labels=cn,
                                     cmap=plt.cm.Blues,
                                     normalize=None)
disp.ax_.set_title('Decision Tree Confusion matrix, without normalization');
```



[23]

```python
# Guassian Naive Bayes Classifier
mod_gnb_all = GaussianNB()
y_pred = mod_gnb_all.fit(X_train, y_train).predict(X_test)
print('The accuracy of the Guassian Naive Bayes Classifier on test data is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test))
```

```
The accuracy of the Guassian Naive Bayes Classifier on test data is 0.933
```

[24]

```python
# Guassian Naive Bayes Classifier with two predictors
mod_gnb = GaussianNB()
y_pred = mod_gnb.fit(X_train[selected_predictors], y_train).predict(X_test[selected_predictors])
print('The accuracy of the Guassian Naive Bayes Classifier with 2 predictors on test data is',"{:.3f}".format(metrics.accuracy_scc
```

```
The accuracy of the Guassian Naive Bayes Classifier with 2 predictors on test data is 0.950
```

[25]

```python
# LDA Classifier
mod_lda_all = LinearDiscriminantAnalysis()
y_pred = mod_lda_all.fit(X_train, y_train).predict(X_test)
print('The accuracy of the LDA Classifier on test data is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))
```

```
The accuracy of the LDA Classifier on test data is 0.983
```

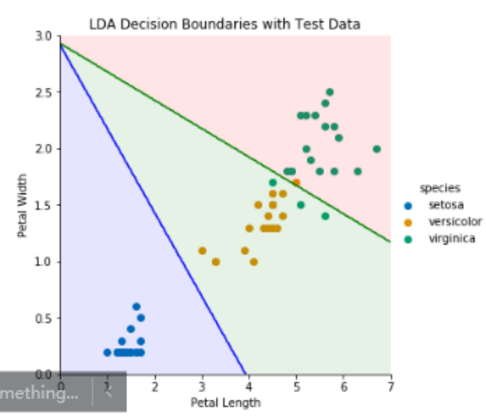
[26]

```python
# LDA Classifier with two predictors
mod_lda = LinearDiscriminantAnalysis()
y_pred = mod_lda.fit(X_train[selected_predictors], y_train).predict(X_test[selected_predictors])
print('The accuracy of the LDA Classifier with two predictors on test data is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_tes
```

```
The accuracy of the LDA Classifier with two predictors on test data is 0.933
```

27]

```python
# LDA with 2 predictors
mod_lda_1 = LinearDiscriminantAnalysis()
y_pred = mod_lda_1.fit(X_train[selected_predictors], y_train_en).predict(X_test[selected_predictors])

N = 300
X = np.linspace(0, 7, N)
Y = np.linspace(0, 3, N)
X, Y = np.meshgrid(X, Y)

g = sns.FacetGrid(test, hue="species", height=5, palette = 'colorblind').map(plt.scatter,"petal_length", "petal_width", ).add_lege
my_ax = g.ax

zz = np.array([mod_lda_1.predict(np.array([[xx,yy]])) for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )
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'))

# Add axis and title
my_ax.set_xlabel('Petal Length')
my_ax.set_ylabel('Petal Width')
my_ax.set_title('LDA Decision Boundaries with Test Data');
```



[28]

```
# QDA Classifier
mod_qda_all = QuadraticDiscriminantAnalysis()
y_pred = mod_qda_all.fit(X_train, y_train).predict(X_test)
print('The accuracy of the QDA Classifier is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))
```

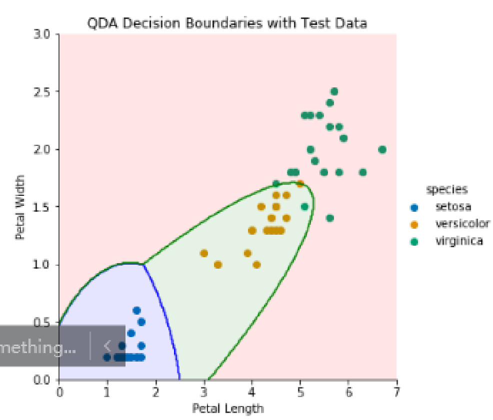
```
The accuracy of the QDA Classifier is 0.983
```

[29]

```
# QDA Classifier with two predictors
mod_qda = QuadraticDiscriminantAnalysis()
y_pred = mod_qda.fit(X_train[selected_predictors], y_train).predict(X_test[selected_predictors])
print('The accuracy of the QDA Classifier with two predictors is',"{:.3f}".format(metrics.accuracy_score(y_pred,y_test)))
```

```
The accuracy of the QDA Classifier with two predictors is 0.967
```

```
# QDA with 2 predictors
mod_qda_1 = QuadraticDiscriminantAnalysis()
y_pred = mod_qda_1.fit(X_train.iloc[:,2:4], y_train_en).predict(X_test.iloc[:,2:4])

N = 300
X = np.linspace(0, 7, N)
Y = np.linspace(0, 3, N)
X, Y = np.meshgrid(X, Y)

g = sns.FacetGrid(test, hue="species", height=5, palette = 'colorblind').map(plt.scatter,"petal_length", "petal_width", ).add_lege
my_ax = g.ax

zz = np.array([mod_qda_1.predict(np.array([[xx,yy]])) for xx, yy in zip(np.ravel(X), np.ravel(Y)) ] )
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('Petal Length')
my_ax.set_ylabel('Petal Width')
my_ax.set_title('QDA Decision Boundaries with Test Data');
```



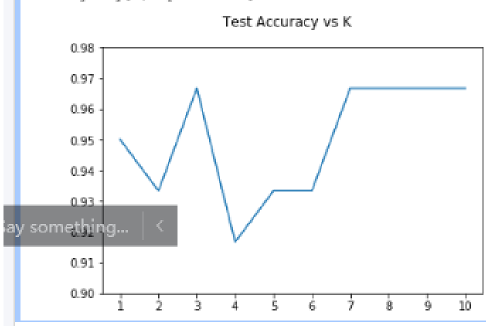
[31]

```
# KNN, first try 5
mod_5nn=KNeighborsClassifier(n_neighbors=5)
mod_5nn.fit(X_train,y_train)
prediction=mod_5nn.predict(X_test)
print('The accuracy of the 5NN Classifier is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
```

```
The accuracy of the 5NN Classifier is 0.933
```

```
# try different k
acc_s = pd.Series(dtype = 'float')
for i in list(range(1,11)):
    mod_knn=KNeighborsClassifier(n_neighbors=i)
    mod_knn.fit(X_train,y_train)
    prediction=mod_knn.predict(X_test)
    acc_s = acc_s.append(pd.Series(metrics.accuracy_score(prediction,y_test)))

plt.plot(list(range(1,11)), acc_s)
plt.suptitle("Test Accuracy vs K")
plt.xticks(list(range(1,11)))
plt.ylim(0.9,0.98);
```

```
/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/cbook/__init__.py:2064: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  x[:, None]
/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/matplotlib/axes/_base.py:250: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is deprecated and will be removed in a future version.  Convert to a numpy array before indexing instead.
  y = y[:, np.newaxis]
```



[33]

```
# SVC with linear kernel
# for SVC, may be impractical beyond tens of thousands of samples
linear_svc = SVC(kernel='linear').fit(X_train, y_train)
prediction=linear_svc.predict(X_test)
print('The accuracy of the linear SVC is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
```

```
The accuracy of the linear SVC is 1.000
```

```
# SVC with polynomial kernel
poly_svc = SVC(kernel='poly', degree = 4).fit(X_train, y_train)
prediction=poly_svc.predict(X_test)
print('The accuracy of the Poly SVC is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
```

```
The accuracy of the Poly SVC is 0.933
```

[35]

```
# Logistic regression
mod_lr = LogisticRegression(solver = 'newton-cg').fit(X_train, y_train)
prediction=mod_lr.predict(X_test)
print('The accuracy of the Logistic Regression is',"{:.3f}".format(metrics.accuracy_score(prediction,y_test)))
```

```
The accuracy of the Logistic Regression is 0.950
```

Use SVM and Random Forest, respectively.

SVM: The "Support Vector Machine" (SVM) is a supervised machine learning technique that can be used for classification and regression tasks. It is, however, largely employed in categorization difficulties. We depict each data item as a point in n-dimensional space (where n is the number of features you have) in the SVM method, with the value of each feature being the value of a certain coordinate**.**

Important Features of Random Forest

1. Diversity- Not all attributes/variables/features are considered while making an individual tree, each tree is different.

2. Immune to the curse of dimensionality- Since each tree does not consider all the features, the feature space is reduced.

3. Parallelization-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. Train-Test split- In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

5. Stability- Stability arises because the result is based on majority voting/ averaging.

2、 Use SVM and Random Forest, respectively.

[12]

```python
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
iris = datasets.load_iris()
```

[13]

```python
# print the label species(setosa, versicolor,virginica)
print(iris.target_names)

# print the names of the four features
print(iris.feature_names)
```

```
['setosa' 'versicolor' 'virginica']
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```python
# print the iris data (top 5 records)
print(iris.data[0:5])

# print the iris labels (0:setosa, 1:versicolor, 2:virginica)
print(iris.target)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
```

```python
# Creating a DataFrame of given iris dataset.
import pandas as pd
data=pd.DataFrame({
    'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target
})
data.head()
```

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

[16]

```python
# Import train_test_split function
from sklearn.model_selection import train_test_split

X=data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features
y=data['species']  # Labels

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% t
```

[17]

```python
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

y_pred=clf.predict(X_test)
```

```python
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

[19]

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

```
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

21]

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

```python
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)

# prediction on test set
y_pred=clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

```python
iris = load_iris()
df_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm','Spe
df_data
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2.0
146	6.3	2.5	5.0	1.9	2.0
147	6.5	3.0	5.2	2.0	2.0
148	6.2	3.4	5.4	2.3	2.0

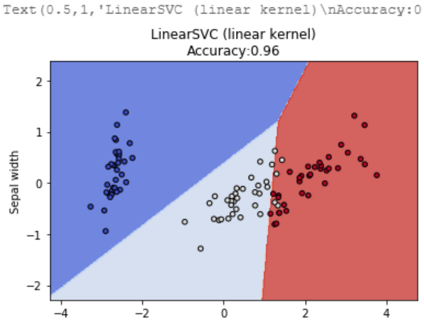
[38]

```
from sklearn.model_selection import train_test_split
X = df_data.drop(labels=['Species'],axis=1).values
y = df_data['Species'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratif

print('train shape:', X_train.shape)
print('test shape:', X_test.shape)
```

```
train shape: (105, 4)
test shape: (45, 4)
```

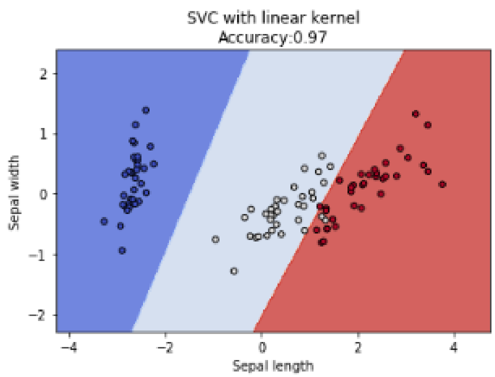
```
from sklearn import svm


linearSvcModel=svm.LinearSVC(C=1, max_iter=10000)

linearSvcModel.fit(train_reduced, y_train)

predicted=linearSvcModel.predict(train_reduced)

accuracy = linearSvcModel.score(train_reduced, y_train)

X0, X1 = train_reduced[:, 0], train_reduced[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(plt, linearSvcModel, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('LinearSVC (linear kernel)'+ '\n' + 'Accuracy:%.2f'%accuracy)
```

```
Text(0.5,1,'LinearSVC (linear kernel)\nAccuracy:0.96')
```



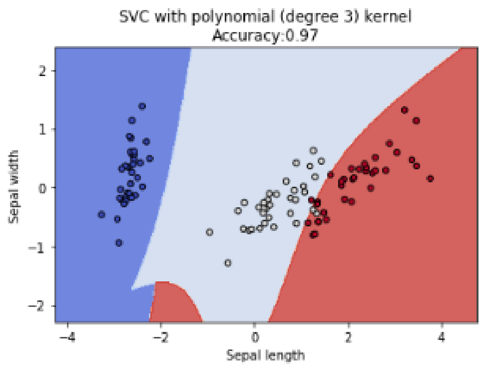
```python
from sklearn import svm


svcModel=svm.SVC(kernel='linear', C=1)

svcModel.fit(train_reduced, y_train)

predicted=svcModel.predict(train_reduced)

accuracy = svcModel.score(train_reduced, y_train)

X0, X1 = train_reduced[:, 0], train_reduced[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(plt, svcModel, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVC with linear kernel'+ '\n' + 'Accuracy:%.2f'%accuracy)
```

```
Text(0.5,1,'SVC with linear kernel\nAccuracy:0.97')
```



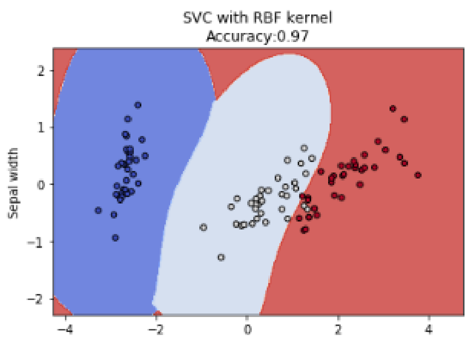
```
from sklearn import svm


polyModel=svm.SVC(kernel='poly', degree=3, gamma='auto', C=1)

polyModel.fit(train_reduced, y_train)

predicted=polyModel.predict(train_reduced)

accuracy = polyModel.score(train_reduced, y_train)

X0, X1 = train_reduced[:, 0], train_reduced[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(plt, polyModel, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVC with polynomial (degree 3) kernel'+ '\n' + 'Accuracy:%.2f'%accuracy)
```

```
Text(0.5,1,'SVC with polynomial (degree 3) kernel\nAccuracy:0.97')
```



```python
from sklearn import svm


rbfModel=svm.SVC(kernel='rbf', gamma=0.7, C=1)

rbfModel.fit(train_reduced, y_train)

predicted=rbfModel.predict(train_reduced)

accuracy = rbfModel.score(train_reduced, y_train)

X0, X1 = train_reduced[:, 0], train_reduced[:, 1]
xx, yy = make_meshgrid(X0, X1)
plot_contours(plt, rbfModel, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVC with RBF kernel'+ '\n' + 'Accuracy:%.2f'%accuracy)
```

```
Text(0.5,1,'SVC with RBF kernel\nAccuracy:0.97')
```



Random Forest,

[48]
```python
from sklearn import datasets
```

[49]
```python
iris = datasets.load_iris()
```

```python
print(iris)
```

```
[6. , 3.4, 4.5, 1.6],
[6.7, 3.1, 4.7, 1.5],
[6.3, 2.3, 4.4, 1.3],
[5.6, 3. , 4.1, 1.3],
[5.5, 2.5, 4. , 1.3],
[5.5, 2.6, 4.4, 1.2],
[6.1, 3. , 4.6, 1.4],
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
```

[52]
```python
import pandas as pd
set = 'data/data.csv'
iris_csv = pd.read_csv(set)
```

[53]
```python
iris_csv.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

ting

```
1 iris_csv.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[54]

```
1 iris_csv.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica

```python
data = pd.DataFrame({'sepal length':iris.data[:,0],
    'sepal width':iris.data[:,1],
    'petal length':iris.data[:,2],
    'petal width':iris.data[:,3],
    'species':iris.target})
data.head()
```

	sepal length	sepal width	petal length	petal width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

[56]

```python
from sklearn.model_selection import train_test_split
X = data[['sepal length', 'sepal width', 'petal length', 'petal width']]  # Features
y = data['species']  # Labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
```

[57]

```python
X.head()
```

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2

[58]
```python
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
clf = RandomForestClassifier(n_estimators=100)
```

```python
clf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[61]
```python
y_pred = clf.predict(X_test)
```

[62]
```python
from sklearn import metrics
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9555555555555556
```

[63]
```python
# Prediction for a single item
clf.predict([[2, 3, 4, 2]])
```

```
array([1])
```

[64]
```python
feature_importance = pd.Series(clf.feature_importances_, index=iris.feature_names).sort_values(ascending=False)
feature_importance
```

```
petal length (cm)    0.449624
petal width (cm)     0.425776
sepal length (cm)    0.100659
sepal width (cm)     0.023941
dtype: float64
```

[65]

```
iris.feature_names
```

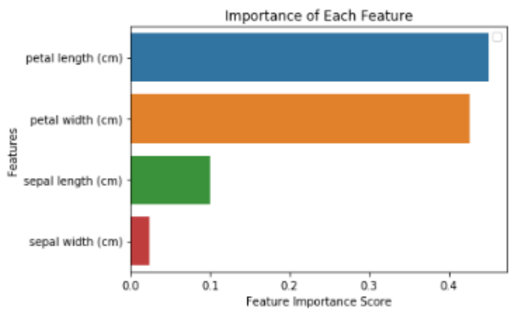
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[66]

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
sns.barplot(x=feature_importance, y=feature_importance.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title('Importance of Each Feature')
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```



[68]

```
# let's remove the feature that's almost irrelevant for the prediction
X = data[['sepal length', 'petal length', 'petal width']]  # Features
y = data['species']  # Labels
```

[69]

```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
from sklearn import metrics
print('Accuracy', metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy 0.9555555555555556
```

Use Grid Search with Cross-Validation to select the best parameters.

Grid search: Grid search is the simplest algorithm for hyperparameter tuning. Basically, we divide the domain of the hyperparameters into a discrete grid. Then, try every combination of values of this grid, calculating some performance metrics using cross-validation. The point of the grid that maximizes the average value in cross-validation, is the optimal combination of values for the hyperparameters.

Cross-Validation: Cross-validation is a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data. Use cross-validation to detect overfitting, failing to generalize a pattern

Corss_validation

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Configures k-NN with 1 neighbor
model = KNeighborsClassifier(n_neighbors=1)

iris = load_iris()
X = iris.data
y = iris.target

# divide the original dataset in 2 equal parts
X1, X2, y1, y2 = train_test_split(X, y, random_state=0, train_size=0.5)

# ajusta e avalia DOIS modelos

y2_model = model.fit(X1, y1).predict(X2)

y1_model = model.fit(X2, y2).predict(X1)

accuracy_score(y1, y1_model), accuracy_score(y2, y2_model)
```

```
(0.96, 0.9066666666666666)
```

[64]

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X, y, cv=10)
print('Scores in each fold: ', scores)
print('Average score: ', scores.mean())
print('Std deviation of scores: ', scores.std())
```

```
Scores in each fold:  [1.   0.93 1.   0.93 0.87 1.   0.87 1.   1.   1.  ]
Average score:  0.96
Std deviation of scores:  0.05333333333333332
```

```
from sklearn.model_selection import LeaveOneOut

scores = cross_val_score(model, X, y, cv=LeaveOneOut())
scores
print('Average score: ', scores.mean())
print('Std deviation of scores: ', scores.std())
```

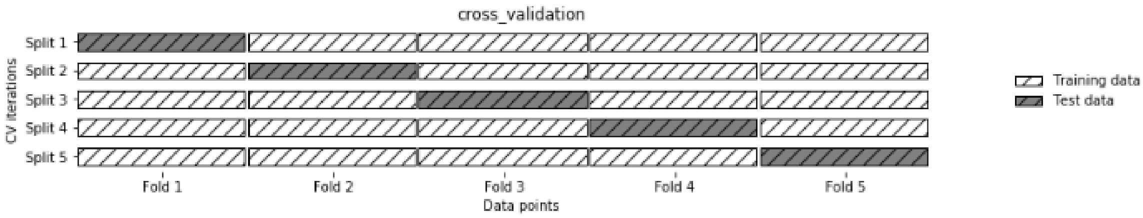
```
Average score:  0.96
Std deviation of scores:  0.19595917942265423
```

[4]

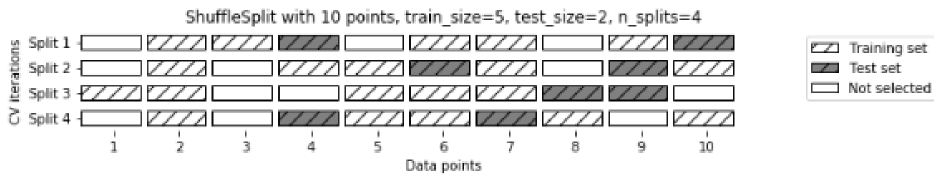
```
from sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# create a synthetic dataset
X, y = make_blobs(random_state=0)
# split data and labels into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# instantiate a model and fit it to the training set
logreg = LogisticRegression().fit(X_train, y_train)
# evaluate the model on the test set
print("Test set score: {:.2f}".format(logreg.score(X_test, y_test)))

```

```
Test set score: 0.88
```

```python
import mglearn
mglearn.plots.plot_cross_validation()
```




```python
mglearn.plots.plot_shuffle_split()
```



`grid search`

[85]

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm, datasets
data = pd.read_csv("data/data.csv")
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa

[86]
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[90]
```
from sklearn.model_selection import GridSearchCV
```

[91]
```
param_grid = {'C':[0.1,1,10,100], 'gamma':[1,0.1,0.01,0.001]}
```

[92]
```
grid = GridSearchCV(SVC(), param_grid, refit = True, verbose=3)
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=1.000, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.913, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=1.000, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.909, total=   0.0s
[CV] C=0.1, gamma=1 ..................................................
[CV] ...................... C=0.1, gamma=1, score=0.955, total=   0.0s
[CV] C=0.1, gamma=0.1 ................................................
[CV] .................... C=0.1, gamma=0.1, score=0.913, total=   0.0s
```

[93]
```
pred_grid = grid.predict(X_test)
```

[94]
```
print(confusion_matrix(y_test, pred_grid))
```

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

[95]
```
print(classification_report(y_test, pred_grid))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

```
# naive grid search implementation
from sklearn.svm import SVC
from sklearn.datasets import load_iris

X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(
      X_train.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

```
Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```

Grid Search with Cross-Validation

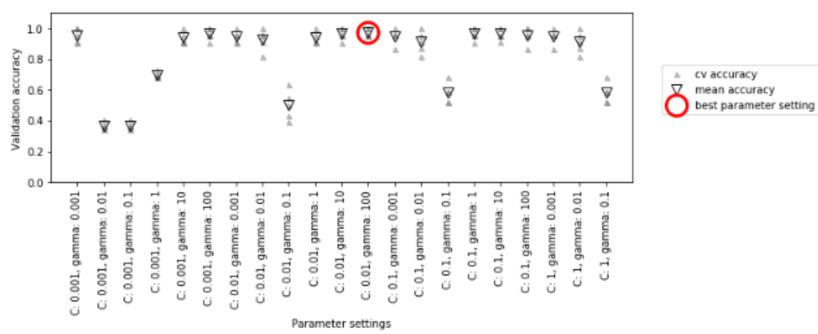
```python
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.datasets import load_iris
import numpy as np
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(
    X_trainval, y_trainval, random_state=1)
print("Size of training set: {}   size of validation set: {}   size of test set:"
      " {}\n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))

best_score = 0

for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the validation set
        score = svm.score(X_valid, y_valid)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}

# rebuild a model on the combined training and validation set,
# and evaluate it on the test set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
test_score = svm.score(X_test, y_test)
print("Best score on validation set: {:.2f}".format(best_score))
print("Best parameters: ", best_parameters)
print("Test set score with best parameters: {:.2f}".format(test_score))
```

```
Size of training set: 84   size of validation set: 28   size of test set: 38

Best score on validation set: 0.96
Best parameters:  {'C': 10, 'gamma': 0.001}
Test set score with best parameters: 0.92
```

```python
from sklearn.datasets import load_iris
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters,
        # train an SVC
        svm = SVC(gamma=gamma, C=C)
        # perform cross-validation
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)
        # compute mean cross-validation accuracy
        score = np.mean(scores)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
# rebuild a model on the combined training and validation set
svm = SVC(**best_parameters)
svm.fit(X_trainval, y_trainval)
```

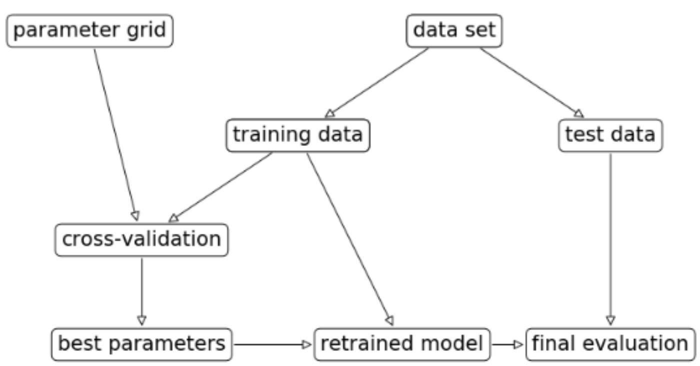
```
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```python
mglearn.plots.plot_cross_val_selection()
```



77]

```python
mglearn.plots.plot_grid_search_overview()
```



[78]

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
print("Parameter grid:\n{}".format(param_grid))
```

```
Parameter grid:
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

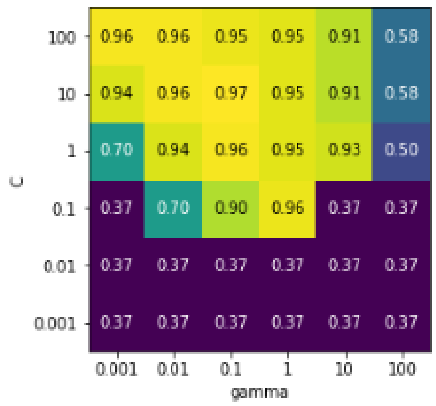
```
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_iris
from sklearn.svm import SVC
grid_search = GridSearchCV(SVC(), param_grid, cv=5,
                           return_train_score=True)
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, random_state=0)
grid_search.fit(X_train, y_train)
print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
import pandas as pd
# convert to Dataframe
results = pd.DataFrame(grid_search.cv_results_)
# show the first 5 rows
display(results.head())
```

```
Test set score: 0.97
Best parameters: {'C': 10, 'gamma': 0.1}
Best cross-validation score: 0.97
Best estimator:
SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	params	split0_test_score	split1_test_score	s
0	0.001134	0.000283	0.000450	0.000129	0.001	0.001	{'C': 0.001, 'gamma': 0.001}	0.347826	0.347826	0
1	0.000839	0.000094	0.000343	0.000028	0.001	0.01	{'C': 0.001, 'gamma': 0.01}	0.347826	0.347826	0
2	0.000913	0.000022	0.000378	0.000010	0.001	0.1	{'C': 0.001, 'gamma': 0.1}	0.347826	0.347826	0
3	0.001001	0.000170	0.000397	0.000059	0.001	1	{'C': 0.001, 'gamma': 1}	0.347826	0.347826	0
4	0.000906	0.000022	0.000374	0.000015	0.001	10	{'C': 0.001, 'gamma': 10}	0.347826	0.347826	0

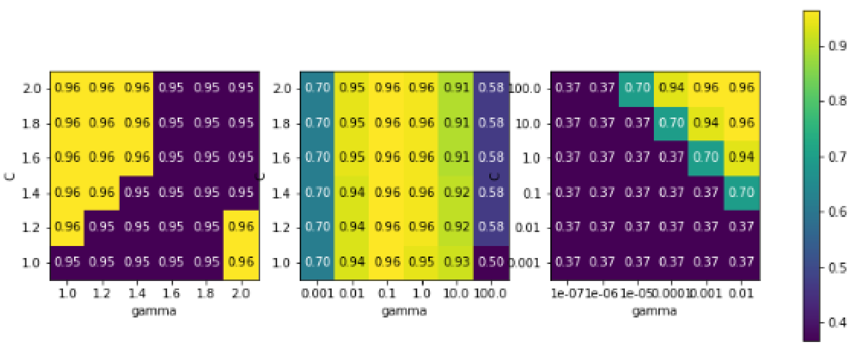
```
scores = np.array(results.mean_test_score).reshape(6, 6)

# plot the mean cross-validation scores
mglearn.tools.heatmap(scores, xlabel='gamma', xticklabels=param_grid['gamma'],
                      ylabel='C', yticklabels=param_grid['C'], cmap="viridis")
```

```
<matplotlib.collections.PolyCollection at 0x7f7374238a10>
```



```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
fig, axes = plt.subplots(1, 3, figsize=(13, 5))

param_grid_linear = {'C': np.linspace(1, 2, 6),
                     'gamma':  np.linspace(1, 2, 6)}

param_grid_one_log = {'C': np.linspace(1, 2, 6),
                      'gamma':  np.logspace(-3, 2, 6)}

param_grid_range = {'C': np.logspace(-3, 2, 6),
                    'gamma':  np.logspace(-7, -2, 6)}

for param_grid, ax in zip([param_grid_linear, param_grid_one_log,
                           param_grid_range], axes):
    grid_search = GridSearchCV(SVC(), param_grid, cv=5)
    grid_search.fit(X_train, y_train)
    scores = grid_search.cv_results_['mean_test_score'].reshape(6, 6)

    # plot the mean cross-validation scores
    scores_image = mglearn.tools.heatmap(
        scores, xlabel='gamma', ylabel='C', xticklabels=param_grid['gamma'],
        yticklabels=param_grid['C'], cmap="viridis", ax=ax)

plt.colorbar(scores_image, ax=axes.tolist())
```

```
<matplotlib.colorbar.Colorbar at 0x7f7361926dd0>
```



Use at least two metrics other than accuracy to evaluate your models.

1- Confusion Matrix

Let’s first make sure we know the basic terminologies used in classification problems before going through the detail of each metrics.

One of the key concepts in classification performance is confusion matrix (AKA error matrix), which is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

Regression Related Metrics

Regression models are another family of machine learning and statistical models, which are used to predict continuous target values. They have a wide range of applications, from house price prediction, E-commerce pricing systems, weather forecasting, stock market prediction, to image super-resolution, feature learning via auto-encoders, and image compression.

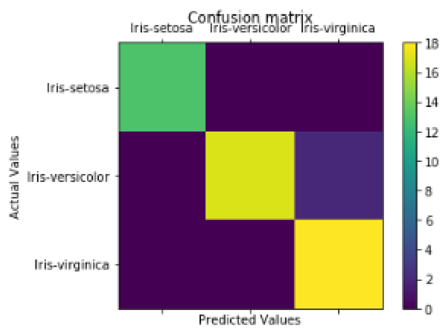
Confusion matrics

```
import warnings
import pandas as pd
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline

#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url)
# df = df.values
X = df.iloc[:,0:4]
y = df.iloc[:,4]
#test size
test_size = 0.33
#generate the same set of random numbers
seed = 7
#Split data into train and test set.
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=seed)
#Train Model
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

```
#Construct the Confusion Matrix
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
cm = confusion_matrix(y_test, pred, labels)
print(cm)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

Execution time: 1seconds639milliseconds   Finished at: 2022-05-25 19:58:12

```
[[13  0  0]
 [ 0 17  2]
 [ 0  0 18]]
```



Accuracy

```
#import modules
import warnings
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import accuracy_score
#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
iris = datasets.load_iris()
# # Create feature matrix
X = iris.data
# Create target vector
y = iris.target
#test size
test_size = 0.33
#generate the same set of random numbers
seed = 7
#cross-validation settings
kfold = model_selection.KFold(n_splits=10, random_state=seed)
#Model instance
model = LogisticRegression()
#Evaluate model performance
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, y, cv=kfold, scoring=scoring)
print('Accuracy -val set: %.2f%% (%.2f)' % (results.mean()*100, results.std()))
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=see
#fit model
model.fit(X_train, y_train)
#accuracy on test set
result = model.score(X_test, y_test)
print("Accuracy - test set: %.2f%%" % (result*100.0))
```

```
Accuracy -val set: 94.67% (0.06)
Accuracy - test set: 92.00%
```

```python
# confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# actual values
actual = [1,0,0,1,0,0,1,0,0,1]
# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]

# confusion matrix
matrix = confusion_matrix(actual,predicted, labels=[1,0])
print('Confusion matrix : \n',matrix)

# outcome values order in sklearn
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy
matrix = classification_report(actual,predicted,labels=[1,0])
print('Classification report : \n',matrix)
```

```
Confusion matrix :
 [[2 2]
 [1 5]]
Outcome values :
 2 2 1 5
Classification report :
               precision    recall  f1-score   support

           1       0.67      0.50      0.57         4
           0       0.71      0.83      0.77         6

    accuracy                           0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10
```

Iris (Plots correlation matrics)

[26]

```
import pandas as pd
pd.options.mode.chained_assignment = None
dataframe = pd.read_csv("data/Iris.csv")

from sklearn.naive_bayes import GaussianNB
from sklearn import svm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Execution time: 9milliseconds Finished at: 2022-05-25 19:15:35

```
# Lets se how our dataframe looks like
dataframe.head()
```

Execution time: 11milliseconds Finished at: 2022-05-25 19:14:08

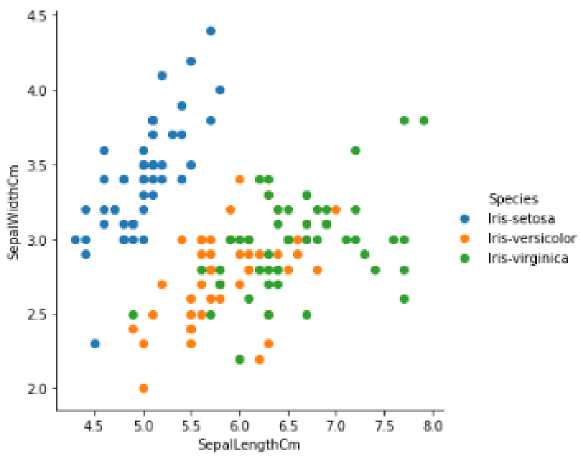
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[24]

```
sns.FacetGrid(dataframe, hue="Species", size=5) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend()
```

Execution time: 417milliseconds Finished at: 2022-05-25 19:14:21

```
/opt/conda/envs/python35-paddle120-env/lib/python3.7/site-packages/seaborn/axisgrid.py:243: UserWarning: The `size
renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.FacetGrid at 0x7fdd74a2e950>
```




```
#In order to rus Naive_Bayes classifier we have to replace the "Species" values
dataframe['Species'].replace("Iris-setosa",1,inplace= True)
dataframe['Species'].replace("Iris-virginica",2,inplace = True)
dataframe['Species'].replace("Iris-versicolor",3,inplace=True)
```

Execution time: 7milliseconds Finished at: 2022-05-25 19:19:21

[33]

```
#Now check if everything was changed properly
dataframe['Species'].unique()
```

Execution time: 7milliseconds Finished at: 2022-05-25 19:19:28

```
array([1, 3, 2])
```

[32]

```
#In order to rus Naive_Bayes classifier we have to replace the "Species" values
dataframe['Species'].replace("Iris-setosa",1,inplace= True)
dataframe['Species'].replace("Iris-virginica",2,inplace = True)
dataframe['Species'].replace("Iris-versicolor",3,inplace=True)
```

Execution time: 7milliseconds Finished at: 2022-05-25 19:19:21

[33]

```
#Now check if everything was changed properly
dataframe['Species'].unique()
```

Execution time: 7milliseconds Finished at: 2022-05-25 19:19:28

```
array([1, 3, 2])
```

[34]

```
X = dataframe.iloc[:, 0:4]
Y = dataframe['Species']
```

Execution time: 5milliseconds Finished at: 2022-05-25 19:19:37

```
# I prefer to use train_test_split for cross-validation
# This peace will prove us if we have overfitting
X_train, X_test, y_train, y_test = train_test_split(
  X, Y, test_size=0.4, random_state=0)
print(" X_train",X_train)
print("x_test",X_test)
print("y_train",y_train)
print("y_test",y_test)
```

Execution time: 21milliseconds Finished at: 2022-05-25 19:19:47

```
 X_train       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm
85    86           6.0           3.4            4.5
30    31           4.8           3.1            1.6
101  102           5.8           2.7            5.1
94    95           5.6           2.7            4.2
64    65           5.6           2.9            3.6
..   ...           ...           ...            ...
9     10           4.9           3.1            1.5
103  104           6.3           2.9            5.6
67    68           5.8           2.7            4.1
117  118           7.7           3.8            6.7
47    48           4.6           3.2            1.4

[90 rows x 4 columns]
```

```
#Train and test model
clf = GaussianNB()
clf = clf.fit(X_train ,y_train)
clf.score(X_test, y_test)
```

Execution time: 14milliseconds Finished at: 2022-05-25 19:20:09

```
0.9833333333333333
```

Regression metrics

[43]

```python
import pandas as pd
# used to read the data set
import numpy as np
# used to do some operations with the arrays
import os
# used handle some files
import matplotlib.pyplot as plt
# used to visualize the data using graphs
import seaborn as sns
# plotting the chart in a single line
```

Execution time: 5milliseconds Finished at: 2022-05-25 19:28:56

```python
df = pd.read_csv("data/Iris.csv")
```

Execution time: 8milliseconds Finished at: 2022-05-25 19:29:26

[45]

```python
df.head(5)
```

Execution time: 13milliseconds Finished at: 2022-05-25 19:29:37

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```python
df = df.drop(columns = ['Id'])
df.head(5)
```

Execution time: 13milliseconds Finished at: 2022-05-25 19:29:46

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

[47]
```
1    df.info()
```

Execution time: 12milliseconds   Finished at: 2022-05-25 19:29:55

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
1    df['Species'].value_counts()
```

Execution time: 9milliseconds   Finished at: 2022-05-25 19:30:03

```
Iris-virginica     50
Iris-versicolor    50
Iris-setosa        50
Name: Species, dtype: int64
```

[49]
```
1    df.isnull().sum()
```

Execution time: 9milliseconds   Finished at: 2022-05-25 19:30:10

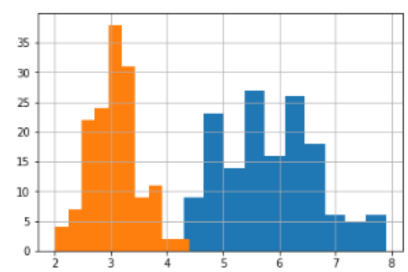
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

[50]
```
1    df['SepalLengthCm'].hist()
2    df['SepalWidthCm'].hist()
```


```
1    df['SepalLengthCm'].hist()
2    df['SepalWidthCm'].hist()
```


Execution time: 206milliseconds   Finished at: 2022-05-25 19:30:19

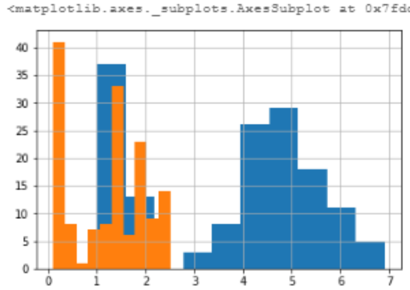
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdd4667e890>
```



[51]
```
1    df['PetalLengthCm'].hist()
2    df['PetalWidthCm'].hist()
```

Execution time: 495milliseconds   Finished at: 2022-05-25 19:30:34

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdd46685bd0>
```



```python
df.corr()
```

Execution time: 12milliseconds   Finished at: 2022-05-25 19:30:46

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

[53]
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
df.head(100)
```

Execution time: 20milliseconds   Finished at: 2022-05-25 19:30:54

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1
98	5.1	2.5	3.0	1.1	1
99	5.7	2.8	4.1	1.3	1

[54]
```python
from sklearn.model_selection import train_test_split
```

Execution time: 4milliseconds   Finished at: 2022-05-25 19:31:04

```python
X = df.drop(columns = ['Species'])
Y = df['Species']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25)
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
```

Execution time: 49milliseconds   Finished at: 2022-05-25 19:31:31

```
LogisticRegression()
```

[58]
```python
print("Accuracy: ", model.score(X_test, Y_test) * 100)
```

Execution time: 8milliseconds   Finished at: 2022-05-25 19:31:41

```
Accuracy:  97.36842105263158
```

Conclusion: I did this project using AI Baidu studio on my computer I also did take a lot of help from recommended books it was a great project for starting I did learn a lot of things. I did every class attentively and also did every homework that's why I did learn so much from this course I believe what I learn from this course I will be able to use this in my future.

Thank you, teacher,