Machine Learning Lab File

Course Code - BCSE3093

School of Computing Science and Engineering

Greater Noida, Uttar Pradesh

Fall 2020 - 2021

Under the Supervision of

Mr. Pratyush Kumar Deka

Submitted by

Name – SAHIL SRIVASTAVA

Admission no. – 18SCSE1010519

Enrollment no. – 18021011747

Section - 2

School of Computing Science & Engineering

Course Name- Machine Learning Lab

Course Code- BCSE3093

S. No.	Title of Lab Experiments
1.	Software setup of machine learning environment
2.	Python revision and introduction to NumPy
3.	Implement Linear Regression with one variable and multiple variable
4.	Implement Logistic Regression to recognize hand-written digits
5.	Implement k-nearest neighbors algorithm
6.	Implement Support Vector Machines to build a spam classifier
7.	Neural network models (supervised)

In [ ]:

```
                                              # EXPERIMENT 2

 #Title : Python revision and introduction to NumPy

 #Theory:Python is a high-level, dynamically typed multiparadigm programming language.
 #Python code isoften said to be almost like pseudocode, since it allows you to
  express
 #very powerful ideas invery few lines of code while being very readable. As an
 example,
 #here is an implementation ofthe classic quicksort algorithm in Python:
```

In [1]:

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)
print(quicksort([3,6,8,10,1,2,1]))
```

```
[1, 1, 2, 3, 6, 8, 10]
```

In [3]:

```
x = 3
print(x + 1)   # Addition
print(x - 1)   # Subtraction
print(x * 2)   # Multiplication
print(x ** 2)  #Exponentiation
```

```
4
2
6
9
```

In [4]:

```
t, f = True, False
print(t andf) # Logical AND;
print(t or f) # Logical OR;
print(not t)   # Logical NOT;
print(t != f) # Logical XOR;
```

```
False
True
False
True
```

In [5]:

```
s = "hello"
print(s.capitalize())  # Capitalize a string
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces
print(s.center(7))     # Center a string, padding with spaces
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with anoth
print('  world '.strip())  # Strip leading and trailing whitespace
```

```
Hello
HELLO
  hello
 hello
```

```
he(ell)(ell)o
world
```

In [6]:

```python
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])
print(xs[-1])
```

```
[3, 1, 2] 2
2
```

In [7]:

```python
xs[2] = 'foo'     # Lists can contain elements of different types
print(xs)
```

```
[3, 1, 'foo']
```

In [8]:

```python
xs.append('bar') # Add a new element to the end of the list
print(xs)
```

```
[3, 1, 'foo', 'bar']
```

In [9]:

```python
x = xs.pop()      # Remove and return the last element of the list
print(x, xs)
```

```
bar [3, 1, 'foo']
```

In [10]:

```python
nums = list(range(5))     # range is a built-in function that creates a list of inte
print(nums)         # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])    # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])     # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])     # Get a slice from the start to index 2 (exclusive); prints "[0
print(nums[:])      # Get a slice of the whole list; prints ["0, 1, 2, 3, 4]"
print(nums[:-1])    # Slice indices can be negative; prints ["0, 1, 2, 3]"
nums[2:4] = [8, 9] # Assign a new sublist to a slice
print(nums)         # Prints "[0, 1, 8, 9, 4]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

In [11]:

```python
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#{}: {}'.format(idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

In [12]:

```python
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
```

```
print(even_squares)
```

```
[0, 4, 16]
```

In [15]:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
```

```
cute
True
```

In [16]:

```
animals = {'cat', 'dog'}
True
False
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals) # prints "False
```

```
True
False
```

In [17]:

```
d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
t = (5, 6)       # Create a tuple
print(type(t))
print(d[t])
print(d[(1, 2)])
```

```
<class 'tuple'>
5
1
```

In [18]:

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
for x in [-1, 0,1]:
    print(sign(x))
```

```
negative
zero
positive
```

In [19]:

```
class Greeter:
    # Constructor
    def_init_(self, name):
        self.name = name  # Create an instance variable
    # Instance method
    def greet(self, loud=False):
        if loud:
          print('HELLO, {}'.format(self.name.upper()))
        else:
          print('Hello, {}!'.format(self.name))
g = Greeter('Fred') # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

```
Hello, Fred!
HELLO, FRED
```

In [1]:

```python
import numpy as np

a = np.array([1, 2, 3])  # Create a rank 1 array
print(type(a), a.shape, a[0], a[1], a[2])
a[0] = 5                 # Change an element of the array
print(a)
b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print(b)

b = np.array([[1,2,3],[4,5,6]])   # Create a rank 2 array
print(b)


a = np.zeros((2,2))  # Create an array of all zeros
print(a)
```

```
<class 'numpy.ndarray'> (3,) 1 2 3
[5 2 3]
[[1 2 3]
 [4 5 6]]
[[1 2 3]
 [4 5 6]]
[[0. 0.]
 [0. 0.]]
```

In [2]:

```python
import numpy as np
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print (b)
```

```
[[2 3]
 [6 7]]
```

In [3]:

```python
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64) # Force a particular datatype
print(x.dtype, y.dtype, z.dtype)
```

```
int32 float64 int64
```

In [4]:

```python
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]],dtype=np.float64)
# Elementwise sum; both produce the array
print(x + y)
print(np.add(x, y))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
```

In [5]:

```
# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

In [6]:

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v
print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [7]:

```
vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each other
print(vv)                # Prints "[[1 0 1]
                         #          [1 0 1]
#          [1 0 1]
#          [1 0 1]]"

y = x + vv  # Add x and vv elementwise
print(y)
```

```
[[1 0 1]
 [1 0 1]
 [1 0 1]
 [1 0 1]]
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [8]:

```
import numpy as np
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v # Add v to each row of x using broadcasting
print (y)
```

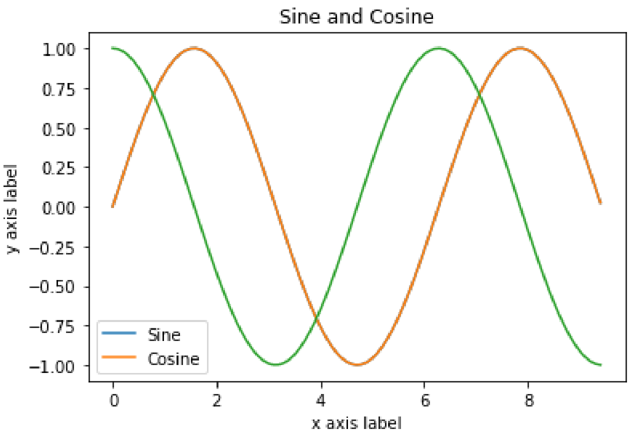
```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

In [9]:

```
import matplotlib.pyplot as plt
%matplotlib inline
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
# Plot the points using matplotlib
plt.plot(x, y)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
```

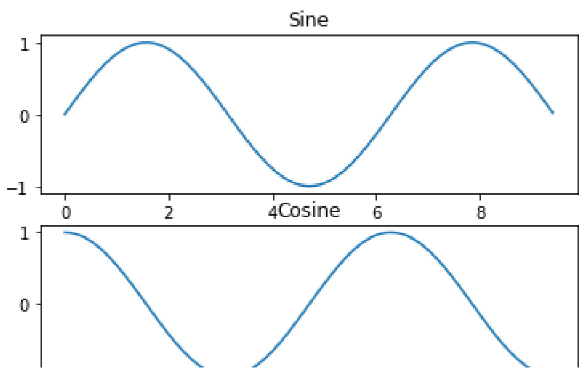
Out[9]:

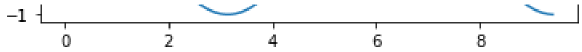
```
<matplotlib.legend.Legend at 0x24712b1fbc8>
```



In [10]:

```
# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)
# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')
# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')
# Show the figure.
plt.show()
```





In [ ]:

In [ ]:

```
                                   #Experiment 3

 #Title : Implement Linear Regression with one variable and multiple variable

 #Theory : linear regression is a linear approach to modeling the relationship
 #between a scalar response (or dependent variable) and one or more explanatory
 #variables (or independent variables). The case of one explanatory variable is
 #called simple linear regression.
```

In [1]:

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(random_state=0)
X = [[ 1,  2,  3],  # 2 samples, 3 features
     [11, 12, 13]]
y = [0, 1] # classes of each sample
clf.fit(X, y)
```

Out[1]:

```
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=0, verbose=0,
                       warm_start=False)
```

In [2]:

```
from sklearn.preprocessing import StandardScaler
X = [[0, 15],
     [1, -10]]
StandardScaler().fit(X).transform(X)
```

Out[2]:

```
array([[-1.,  1.],
       [ 1., -1.]])
```

In [3]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# create a pipeline object
pipe = make_pipeline(
    StandardScaler(),
    LogisticRegression(random_state=0)
)

# load the iris dataset and split it into train and test sets
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# fit the whole pipeline
pipe.fit(X_train, y_train)


# we can now use it like any other estimator
accuracy_score(pipe.predict(X_test),  y_test)
```

Out[3]:

```
0.9736842105263158
```

In [4]:

```python
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_validate

X, y = make_regression(n_samples=1000, random_state=0)
lr = LinearRegression()

result = cross_validate(lr, X, y) # defaults to 5-fold CV
result['test_score'] # r_squared score is high because dataset is easy
```

Out[4]:

```
array([1., 1., 1., 1., 1.])
```

In [ ]:

```python

```

In [ ]:

```
                                              #Experiment 4

    #Title : Implement Logistic Regression to recognize hand-written digits

    #Theory : Logistic Regression is an algorithm that is admirably suited for
    #discovering the link between features or cues and some particular outcome.
    #Logistic regression is one of the most important analytic tools in the social and natural sci
ences.
```

In [1]:

```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

reg.coef_
```

Out[1]:

```
array([0.5, 0.5])
```

In [2]:

```
from sklearn import linear_model
reg =linear_model.Ridge(alpha=.5)
reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1,1])

reg.coef_

reg.intercept_
```

Out[2]:

```
0.1363636363636364
```

In [3]:

```
import numpy as np
from sklearn import linear_model
reg = linear_model.RidgeCV(alphas=np.logspace(-6, 6, 13))
reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1,1])


reg.alpha_
```

Out[3]:

```
0.01
```

In [4]:

```
from sklearn import linear_model
reg = linear_model.Lasso(alpha=0.1)
reg.fit([[0, 0], [1, 1]], [0,1])

reg.predict([[1, 1]])
```

Out[4]:

```
array([0.8])
```

In [5]:

```
#First example of OLS
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)
reg.score(X, y)

reg.coef_

reg.intercept_

reg.predict(np.array([[3, 5]]))
```

Out[5]:

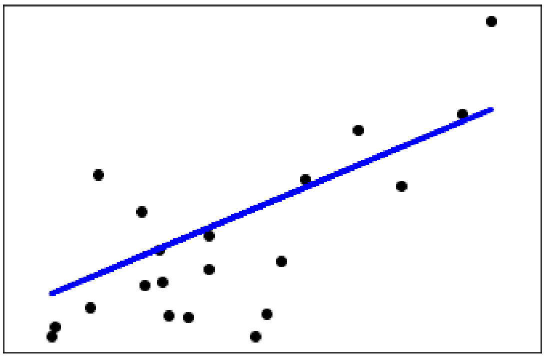
```
array([16.])
```

In [6]:

```
#Linear Regression with in-built Diabetes dataset
print(_doc__)


# Code source: Jaques Grobler
# License: BSD 3 clause


import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```

```
Automatically created module for IPython interactive environment
Coefficients:
 [938.23786125]
```

```
Mean squared error: 2548.07
Coefficient of determination: 0.47
```



```
In [ ]:
```

In [ ]:

```
                                                  #Experiment 5

    #Title : Implement k-nearest neighbors algorithm

    # Theory: K nearest neighbors is a simple algorithm that stores
    #all available cases and classifies new cases based on a similarity measure
```

In [1]:

```
#1.Study Logistic Regression in Scikit-learn
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0).fit(X, y)
clf.predict(X[:2, :])

clf.predict_proba(X[:2, :])


clf.score(X, y)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:940: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[1]:

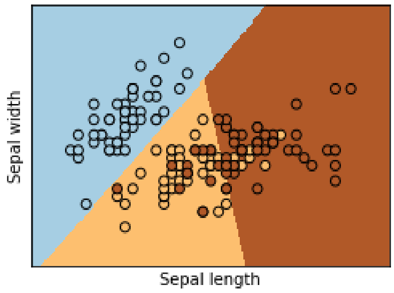
```
0.9733333333333334
```

In [2]:

```
#2. Build a classifier using Logistic Regression with in-built Iris dataset
print(_doc__)



import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
Y = iris.target

logreg = LogisticRegression(C=1e5)

# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)
```

```
# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

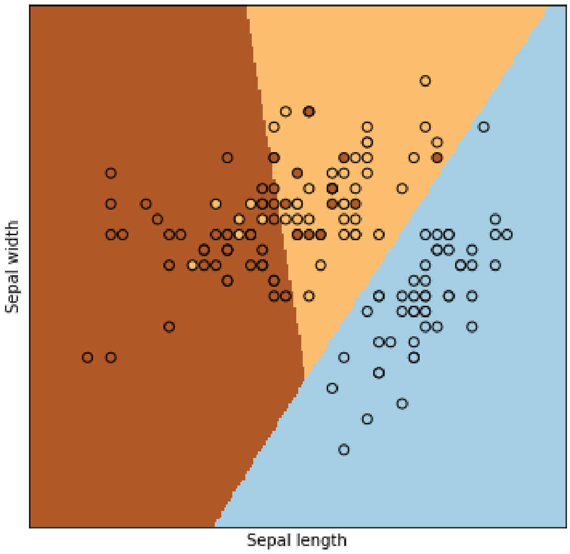
```
Automatically created module for IPython interactive environment
```



In [3]:

```
#3.Change features from sepal length and width to petal length and width. Build the classifier aga
in and discuss the output.
print(_doc__)



import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
Y = iris.target

logreg = LogisticRegression(C=1e5)

# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(6, 6))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.max(), xx.min())
plt.ylim(yy.max(), yy.min())
plt.xticks(())
plt.yticks(())

plt.show()
```

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



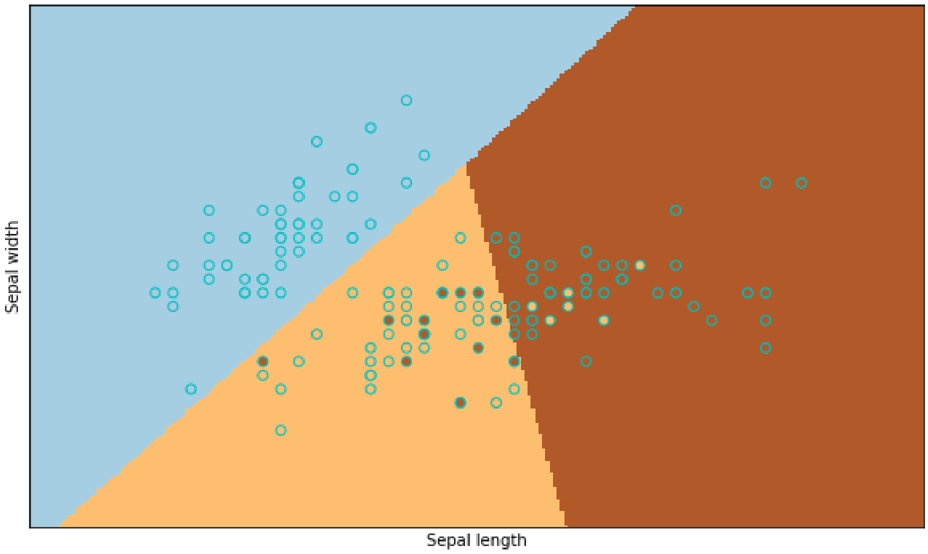
In [ ]:

```
sepal width and sepal length get changed also graph get swapped.
```

In [4]:

```
#4.Consider all the features and build the classifier again and discuss the output.
print(_doc__)



import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features.
Y = iris.target

logreg = LogisticRegression(C=1e5)

# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X, Y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - .7, X[:, 0].max() + .7
y_min, y_max = X[:, 1].min() - .7, X[:, 1].max() + .7
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(10, 6))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='c', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

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



In [ ]:

```
length and width get changed. Also colour of points get changed. Spaces between points also get cha
nged.
```

In [ ]:

```
                                        #Experiment6

    #Title: Implement Support Vector Machines to build a spam classifier

    #Theory:  A support vector machine is a supervised machine learning model used for
classification.
```

In [1]:

```
#Nearest Neighbors Classification in Scikit-learn
from sklearn.neighbors import NearestNeighbors
import numpy as np
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
nbrs = NearestNeighbors(n_neighbors=2, algorithm='ball_tree').fit(X)
distances, indices = nbrs.kneighbors(X)
indices







distances
```

Out[1]:

```
array([[0.        , 1.        ],
       [0.        , 1.        ],
       [0.        , 1.41421356],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.        , 1.41421356]])
```

In [2]:

```
nbrs.kneighbors_graph(X).toarray()
```

Out[2]:

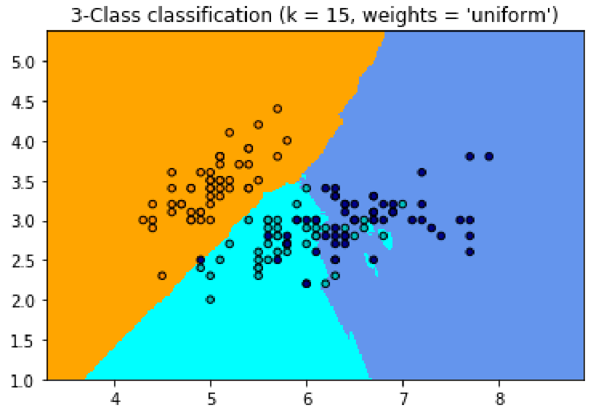
```
array([[1., 1., 0., 0., 0., 0.],
       [1., 1., 0., 0., 0., 0.],
       [0., 1., 1., 0., 0., 0.],
       [0., 0., 0., 1., 1., 0.],
       [0., 0., 0., 1., 1., 0.],
       [0., 0., 0., 0., 1., 1.]])
```

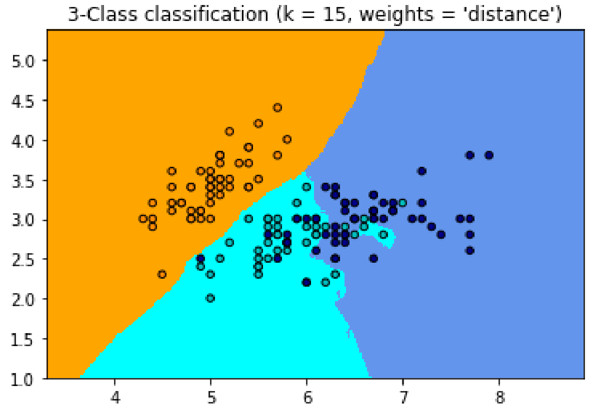
In [3]:

```
#Build a classifier using k-Nearest Neighbors Classifier with in-built Iris dataset and plot the d
ecision boundaries of each class

print(_doc__)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = .02  # step size in the mesh
```

```
# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=20)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```

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



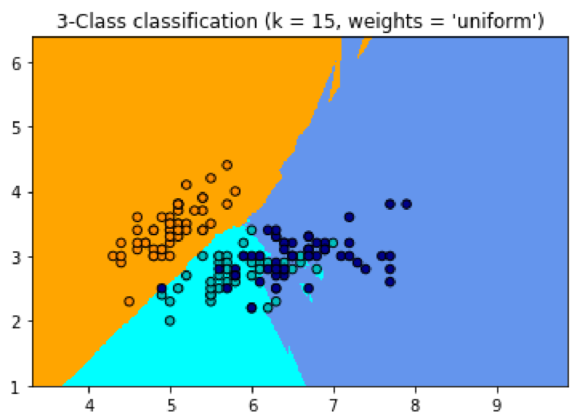


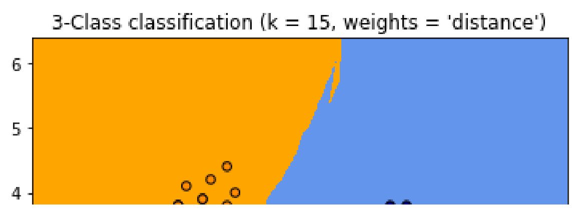
In [4]:

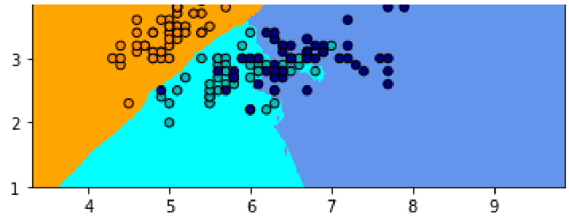
```
#Change features from sepal length and width to petal length and width. Build the classifier again
and discuss the output.
print(_doc__)
```

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkorange', 'c', 'darkblue'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 2
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 2
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max,h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=30)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```

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



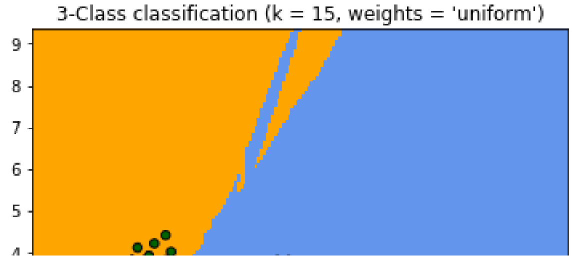


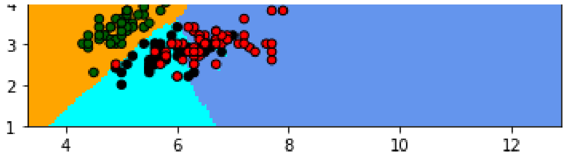


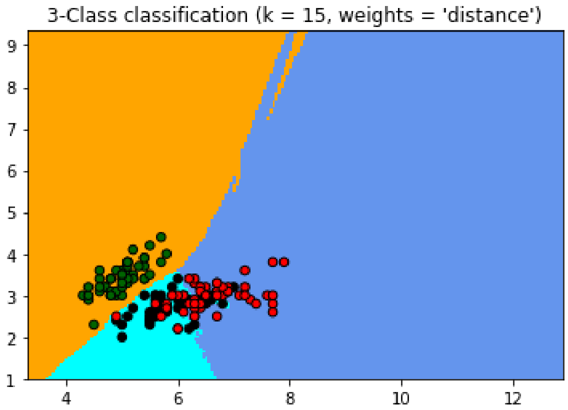
In [5]:

```
#Consider all the features and build the classifier again and discuss the output.
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

n_neighbors = 15

# import some data to play with
iris = datasets.load_iris()

# we only take the first two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = iris.data[:, :2]
y = iris.target

h = .05  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['orange', 'cyan', 'cornflowerblue'])
cmap_bold = ListedColormap(['darkgreen', 'black', 'red'])

for weights in ['uniform', 'distance']:
    # we create an instance of Neighbours Classifier and fit the data.
    clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
    clf.fit(X, y)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 5
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=30)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("3-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, weights))

plt.show()
```

Automatically created module for IPython interactive environment



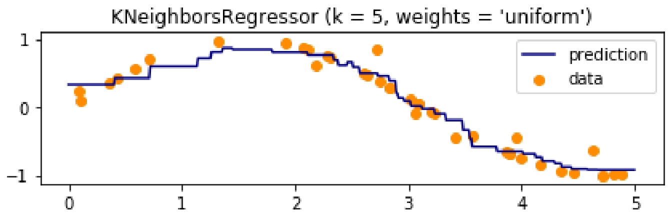


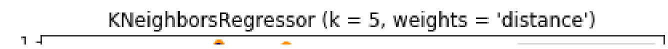


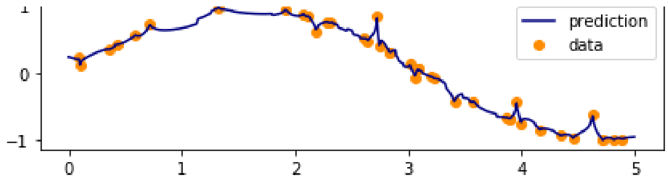
In [6]:

```
#Demonstrate the resolution of a regression problem using a k-Nearest Neighbor and the interpolati
on of the target
print(_doc__)


# Generate sample data
import numpy as np
import matplotlib.pyplot as plt
from sklearn import neighbors

np.random.seed(0)
X = np.sort(5 * np.random.rand(40, 1), axis=0)
T = np.linspace(0, 5, 500)[:, np.newaxis]
y = np.sin(X).ravel()

# Add noise to targets
y[::5] += 1 * (0.5 - np.random.rand(8))

# #############################################################################
# Fit regression model
n_neighbors = 5

for i, weights in enumerate(['uniform', 'distance']):
    knn = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
    y_ = knn.fit(X, y).predict(T)

    plt.subplot(2, 1, i + 1)
    plt.scatter(X, y, color='darkorange', label='data')
    plt.plot(T, y_, color='navy', label='prediction')
    plt.axis('tight')
    plt.legend()
    plt.title("KNeighborsRegressor (k = %i, weights = '%s')" % (n_neighbors,
                                                                weights))

plt.tight_layout()
plt.show()
```

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







prediction
data
1
0
−1
0
1
2
3
4
5

In [ ]:

```
Neural network models (supervised)
```

In [1]:

```
#MLP trains on two arrays: array X of size (n_samples, n_features),
#which holds the training samples represented as floating point feature vectors;
#and array y of size (n_samples,), which holds the target values (class labels) for the tra

from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.]]
y = [0, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)

clf.fit(X, y)
```

Out[1]:

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(5, 2), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=200,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='lbfgs',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
```

In [2]:

```
#After fitting (training), the model can predict labels for new samples:

clf.predict([[2., 2.], [-1., -2.]])
```

Out[2]:

```
array([1, 0])
```

In [3]:

```
#MLP can fit a non-linear model to the training data.
#clf.coefs_ contains the weight matrices that constitute the model parameters:

[coef.shape for coef in clf.coefs_]
```

Out[3]:

```
[(2, 5), (5, 2), (2, 1)]
```

In [4]:

```
#MLP trains using Backpropagation. More precisely, it trains using some form of gradient de
#and the gradients are calculated using Backpropagation.
#For classification, it minimizes the Cross-Entropy loss function, giving a vector of proba

clf.predict_proba([[2., 2.], [1., 2.]])
```

Out[4]:

```
array([[1.96718015e-04, 9.99803282e-01],
       [1.96718015e-04, 9.99803282e-01]])
```

In [5]:

```
#MLPClassifier supports multi-class classification by applying Softmax as the output functi

X = [[0., 0.], [1., 1.]]
y = [[0, 1], [1, 1]]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(15,), random_state=1)

clf.fit(X, y)


clf.predict([[1., 2.]])

clf.predict([[0., 0.]])
```

Out[5]:

```
array([[0, 1]])
```

In [ ]:

In [ ]:

```
#The aim is to build a classification model to detect diabetes. We will be using the diabet
#dataset (Kaggle) which contains 768 observations and 9 variables
```

In [1]:

```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
# Import necessary modules
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import r2_score
```

In [2]:

```
df = pd.read_csv('D:\diabetes.csv')
print(df.shape)
df.describe().transpose()
```

(768, 9)

Out[2]:

	count	mean	std	min	25%	50%	75%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000

In [3]:

```
target_column = ['diabetes']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe().transpose()
```

Out[3]:

	count	mean	std	min	25%	50%	75%
Pregnancies	768.0	0.226180	0.198210	0.000000	0.058824	0.176471	0.352941
Glucose	768.0	0.607510	0.160666	0.000000	0.497487	0.587940	0.704774
BloodPressure	768.0	0.566438	0.158654	0.000000	0.508197	0.590164	0.655738
SkinThickness	768.0	0.207439	0.161134	0.000000	0.000000	0.232323	0.323232
Insulin	768.0	0.094326	0.136222	0.000000	0.000000	0.036052	0.150414
BMI	768.0	0.476790	0.117499	0.000000	0.406855	0.476900	0.545455
DiabetesPedigreeFunction	768.0	0.194990	0.136913	0.032231	0.100723	0.153926	0.258781
Age	768.0	0.410381	0.145188	0.259259	0.296296	0.358025	0.506173
Outcome	768.0	0.348958	0.476951	0.000000	0.000000	0.000000	1.000000

In [4]:

```
X, y = df.drop('Outcome', axis=1), df['Outcome']
print(X.shape, y.shape)

# type(X)   # pandas.core.frame.DataFrame
# type(y)   # pandas.core.series.Series
```

```
(768, 8) (768,)
```

In [5]:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='relu', solver='adam', max_iter=
mlp.fit(X,y)
predict_train = mlp.predict(X)
predict_test = mlp.predict(X)

from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y,predict_train))
print(classification_report(y,predict_train))

print(confusion_matrix(y,predict_test))
print(classification_report(y,predict_test))
```

```
[[436  64]
 [ 95 173]]
              precision    recall  f1-score   support

         0.0       0.82      0.87      0.85       500
         1.0       0.73      0.65      0.69       268

    accuracy                           0.79       768
   macro avg       0.78      0.76      0.77       768
weighted avg       0.79      0.79      0.79       768

[[436  64]
 [ 95 173]]
              precision    recall  f1-score   support

         0.0       0.82      0.87      0.85       500
         1.0       0.73      0.65      0.69       268

    accuracy                           0.79       768
   macro avg       0.78      0.76      0.77       768
weighted avg       0.79      0.79      0.79       768
```

In [ ]: