Python Tutorial for Applications of SCIKIT Library

SCIKIT-LEARNS Liabrary

https://scikit-learn.org/0.17/index.html

Installation: https://scikit-learn.org/0.17/install.html

Clustring, Regression and Classifications Task:

https://scikit-learn.org/0.17/supervised_learning.html#supervised-learning

Datasets for practical applications of NNs

- link: https://scikit-learn.org/stable/datasets.html

In [1]:
```
# Optical recognition of handwritten digits dataset  By C. Kaynak (1998)
import pandas as pd
from sklearn.datasets import load_digits
digits = load_digits()
digits.keys()
```

Out[1]:
```
dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])
```

In [2]:
```
print(digits.data.shape)

print(digits.target.shape)
```

```
(1797, 64)
(1797,)
```

In [5]:
```
df=pd.DataFrame(digits.data)
df
# print(df.head())
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	54	55	56	57	58	59	60	61	62	63
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0

1797 rows × 64 columns

In [25]:
```
print(df.shape)
```

```
(1797, 64)
```

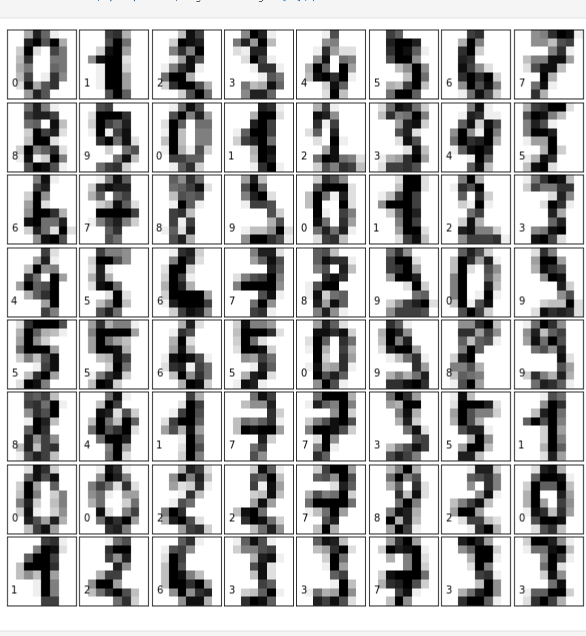
In [7]:
```
df['target']=digits.target

df
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	target
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	2.0	14.0	15.0	9.0	0.0	0.0	9
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	16.0	14.0	6.0	0.0	0.0	0
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	2.0	9.0	13.0	6.0	0.0	0.0	8
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	5.0	12.0	16.0	12.0	0.0	0.0	9
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	0.0	0.0	1.0	8.0	12.0	14.0	12.0	1.0	0.0	8

1797 rows × 65 columns

In [9]:
```
from matplotlib import pyplot as plt
fig = plt.figure(figsize=(8, 8))  # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(digits.images[i], cmap=plt.cm.binary, interpolation='nearest')
    # label the image with the target value
    ax.text(0, 6, str(digits.target[i]))
```



In [11]:
```
from sklearn.model_selection import train_test_split

# split the data into training and validation sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,test_size=0.20)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1437, 64)
(360, 64)
(1437,)
(360,)
```

Supervised and Unsupervised Algorithms

Multilayer Perceptron(MLP) Neural networks Supervised Algorithm

In [12]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
# from sklearn.datasets import make_classification
# X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
# ...                                                    random_state=1)
# mlpmodel = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
mlpmodel = MLPClassifier(random_state=1, max_iter=200)
mlpmodel =mlpmodel.fit(X_train, y_train)
prob_pred=mlpmodel.predict_proba(X_test[:5])
print(prob_pred)
out_pred_class=mlpmodel.predict(X_test[:5])
# out_pred_class=mlpmodel.predict(X_test[:10,:])
print(out_pred_class)
avg_test_acc=mlpmodel.score(X_test, y_test)
print(avg_test_acc)
out_pred=mlpmodel.predict(X_test)
acc=accuracy_score(out_pred,y_test)
print(acc)
```

```
[[3.88809302e-07 5.28309533e-08 6.70864727e-11 2.74182067e-15
  9.99902574e-01 5.83172744e-10 2.26150613e-07 9.64501872e-05
  3.07817216e-07 1.15141308e-12]
 [1.01693203e-04 4.86984475e-04 9.99287179e-01 7.37415096e-06
  1.29030976e-07 3.62778126e-05 6.63417781e-07 1.60029189e-08
  7.89740029e-05 7.09284608e-07]
 [4.99262718e-09 4.31282266e-08 1.56765513e-06 9.97600906e-01
  3.32180874e-12 4.89933562e-07 3.77720280e-12 2.34881042e-03
  1.51292845e-06 4.66649152e-05]
 [4.93029658e-06 1.17982004e-04 1.35058894e-04 1.04233026e-05
  2.15739940e-07 1.81345240e-05 1.34186483e-03 1.73859938e-08
  9.98260281e-01 1.11092124e-04]
 [1.23376091e-11 1.03996294e-08 3.53935626e-09 1.38576311e-07
  1.42652922e-10 3.75303408e-07 6.15585801e-15 9.99999435e-01
  1.53389501e-09 3.57431690e-08]]
[4 2 3 8 7]
0.9638888888888889
0.9638888888888889
```

SVM Supervised Algorithm

What is Kernel?

- A kernel is a function used in SVM for helping to solve problems
- They provide shortcuts to avoid complex calculations.
- The good thing about kernel is that we can go to higher dimensions and perform smooth calculations with the help of it.

Working of Kernel Functions

- Kernels are a way to solve non-linear problems with the help of linear classifiers.
- This is known as the kernel trick method.
- The kernel functions are used as parameters in the SVM codes.
- They help to determine the shape of the hyperplane and decision boundary.
- The value can be any type of kernel from linear to polynomial.
- If the value of the kernel is linear then the decision boundary would be linear and two-dimensional.
- These kernel functions also help in giving decision boundaries for higher dimensions.
- We do not need to do complex calculations.
- The kernel functions do all the hard work.
- We just have to give the input and use the appropriate kernel. Also, we can solve the overfitting problem in SVM using the kernel functions.
- Overfitting happens when there are more feature sets than sample sets in the data. We can solve the problem by either increasing the data or by choosing the right kernel. ## Kernel types
- Polynomial Kernel, Gaussian Kernel, Radial Basis Function (RBF), Laplace RBF Kernel, Sigmoid Kernel, Linear Kernel etc.
- Link for more details about kernel types https://data-flair.training/blogs/svm-kernel-functions/ ### What is Support vectors
- Support vectors are data points that defines the position and the margin of the hyperplane.We call them "support" vectors, because these are the representative data points of the classes, if we move one of them, the position and/or the margin will change. ### Basic Steps in SVM
- Select two hyperplanes (in 2D) which separates the data with no points between them (red lines)
- Maximize their distance (the margin)
- The average line (here the line half way between the two red lines) will be the decision boundary ### This is very nice and easy, but finding the best margin, the optimization problem is not trivial (it is easy in 2D, when we have only two attributes, but what if we have N dimensions with N a very big number) ### To solve the optimization problem, we use the Lagrange Multipliers.

In [13]:
```
## Linear Kernel
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC
model1=SVC(kernel='linear',random_state=0, probability=True)

model1.fit(X_train,y_train)

y_pred_1=model1.predict(X_test)
print(len(y_pred_1))
print("Predicted classes of hand written digits")
print(y_pred_1)
print(" Original classes of hand written digits")
print(y_test)


print("Model Score of Kernal(linear) :", model1.score(X_test,y_test))
acc=accuracy_score(y_pred_1,y_test)
print(acc)
```

```
360
Predicted classes of hand written digits
[4 2 3 8 7 1 9 2 4 8 0 0 4 9 9 6 4 5 2 0 5 9 8 4 8 1 3 5 1 1 6 8 7 8 9 9 9
 1 0 8 1 7 5 1 8 3 3 7 7 6 0 7 8 4 5 4 5 3 3 6 0 3 1 1 8 7 8 5 8 3 3 1 4 3
 3 3 3 3 9 9 2 1 7 4 9 6 2 7 3 4 3 4 4 7 5 4 4 6 6 6 5 5 2 8 1 8 1 7 7 8 5
 8 8 5 1 1 3 0 8 1 1 8 1 5 3 3 6 1 7 5 7 9 7 5 6 5 2 0 3 4 2 4 6 3 6 0 7 8
 9 5 0 2 1 1 0 7 7 3 1 0 9 8 3 9 6 1 8 9 3 6 9 2 2 9 4 4 1 1 2 6 2 7 6 9 0
 9 0 3 3 9 9 2 6 0 9 0 5 9 7 8 5 5 4 5 6 8 0 1 0 2 3 7 4 8 3 3 3 0 7 1 9 3
 4 1 1 7 0 2 0 9 0 5 8 7 4 0 5 8 8 2 7 7 9 8 3 1 9 4 0 3 4 6 1 3 3 8 7 1 5
 8 7 3 6 9 0 1 6 9 3 2 9 5 9 5 5 1 4 5 0 0 3 6 2 6 1 0 7 0 1 4 5 2 5 8 3 6
 1 7 4 9 3 3 4 1 7 8 4 2 6 0 6 3 7 6 7 2 5 2 9 1 4 7 8 1 0 2 9 5 7 6 7 6 2
 9 8 1 2 4 2 4 8 8 2 3 8 3 9 6 2 2 3 0 3 6 9 1 7 7 7 1]
 Original classes of hand written digits
[4 2 3 8 7 1 9 2 4 8 0 0 4 9 9 6 4 5 2 0 5 9 8 4 8 1 3 5 1 1 6 8 7 8 9 9 9
 1 0 8 1 7 5 1 8 3 3 7 7 6 0 7 8 4 5 4 5 3 3 6 0 3 1 1 8 7 8 5 8 3 3 1 4 3
 3 3 3 3 9 9 2 1 7 4 9 6 2 7 3 4 3 4 4 7 5 4 4 6 6 6 5 5 2 8 1 8 8 7 7 8 5
 8 8 5 1 1 3 0 8 1 1 8 1 5 3 3 6 1 7 5 7 9 7 5 6 5 2 0 3 4 2 4 6 3 6 0 7 8
 9 5 0 2 1 1 0 9 7 3 1 0 9 8 3 9 6 1 8 9 3 6 9 2 2 9 4 4 1 1 2 6 2 7 6 9 0
 9 0 3 3 9 9 2 6 0 9 0 5 9 7 8 5 5 4 5 6 8 0 8 0 2 3 7 4 8 3 8 3 0 7 1 9 3
 4 1 8 7 0 2 0 9 0 9 8 7 4 0 5 8 3 2 7 7 9 8 3 1 9 4 0 3 4 6 1 3 3 8 7 1 5
 8 7 3 6 9 0 1 6 9 3 2 9 5 9 5 5 8 4 5 0 0 3 6 2 6 1 0 7 0 1 4 5 2 5 8 3 6
 1 7 4 9 3 3 4 1 7 8 4 2 6 0 6 3 7 6 7 2 5 2 9 1 4 7 8 1 0 2 9 5 7 6 7 6 2
 9 8 1 2 4 2 4 8 8 2 3 8 3 9 6 2 2 3 0 3 6 9 1 7 7 7 1]
Model Score of Kernal(linear) : 0.9777777777777777
0.9777777777777777
```

In [44]:
```
## Sigmoid Kernel  function
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC

model2=SVC(kernel='sigmoid',random_state=0, probability=True)

model2.fit(X_train,y_train)

y_pred_2=model2.predict(X_test)
print("Predicted classes of hand written digits")
print(y_pred_2)
print(" Original classes of hand written digits")
print(y_test)


print("Model Score of Kernal(rbf) :", model2.score(X_test,y_test))
acc=accuracy_score(y_pred_2,y_test)
print(acc)
```

```
Predicted classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 3 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 9 0 4 2 7 6 1 0 5 6 1 0 8 3 5 8 6 4 0 7
 6 9 1 2 8 6 5 8 4 9 9 3 6 8 5 4 9 1 6 9 8 1 1 1 3 1 4 7 2 6 1 9 1 5 1 9 4
 7 6 4 2 8 0 4 8 7 8 7 7 1 3 6 7 9 9 0 3 7 5 6 6 0 6 8 1 7 8 9 2 9 8 7 6 1
 5 2 5 7 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 3 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 1 8 2 2 2 9 3 8 3 0 7 5 3 4 1 7 7 2 1 4 5 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 3 9 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 4 1 5 6 3 1 2 2 8 8 2 2 2 2 9 3 8 8 0
 0 7 1 9 4 2 5 1 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 4 1 4 8 9 0 2 6 9 3 1 4 4
 5 8 5 4 4 0 5 4 7 7 6 7 3 9 4 0 1 9 1 1 4 7 8 1 8 8 2]
 Original classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 8 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 8 0 4 2 7 6 1 0 5 6 1 0 8 5 3 8 6 4 0 7
 6 9 7 2 8 6 5 8 4 1 9 3 6 8 5 4 9 8 6 9 8 1 1 1 3 1 4 7 2 6 8 9 2 5 1 9 4
 7 6 4 2 2 0 4 8 7 1 7 7 1 3 6 7 9 9 0 3 8 5 6 6 0 6 1 1 7 8 9 2 9 8 7 6 1
 5 2 5 1 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 9 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 4 8 2 2 2 9 3 8 3 0 7 5 8 4 1 7 7 2 1 4 9 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 8 1 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 5 1 5 4 3 1 2 2 8 8 2 2 2 2 9 3 7 8 0
 0 1 1 1 4 8 5 4 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 0 1 6 8 9 0 2 6 8 3 1 4 1
 5 8 7 4 0 0 5 4 7 7 6 7 3 1 4 0 1 9 8 1 4 7 8 1 8 9 2]
Model Score of Kernal(rbf) : 0.9
0.9
```

In [45]:
```
## Radial Basis function Kernel
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC

model3=SVC(kernel='rbf',random_state=0, probability=True)

model3.fit(X_train,y_train)

y_pred_3=model3.predict(X_test)
print("Predicted classes of hand written digits")
print(y_pred_3)
print(" Original classes of hand written digits")
print(y_test)


print("Model Score of Kernal(rbf) :", model3.score(X_test,y_test))
acc=accuracy_score(y_pred_3,y_test)
print(acc)
```

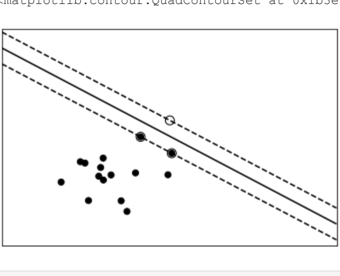
```
Predicted classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 8 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 8 0 4 2 7 6 1 0 5 6 1 0 8 9 3 8 6 4 0 7
 6 9 7 2 8 6 5 8 4 1 9 3 6 8 5 4 9 1 6 9 8 1 1 1 3 1 4 7 2 6 8 9 2 5 1 9 4
 7 6 4 2 2 0 4 8 7 1 7 7 1 3 6 7 9 9 0 3 8 5 6 6 0 6 1 1 7 8 9 2 9 8 7 6 1
 5 2 5 1 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 9 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 4 8 2 2 2 9 3 8 3 0 7 5 8 4 1 7 7 2 1 4 9 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 8 1 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 5 1 5 4 3 1 2 2 8 8 2 2 2 2 9 3 7 8 0
 0 1 1 1 4 8 5 4 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 0 1 6 8 9 0 2 6 8 3 1 4 1
 5 8 7 4 0 0 5 4 7 7 6 7 3 1 4 0 1 9 8 1 4 7 8 1 8 9 2]
 Original classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 8 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 8 0 4 2 7 6 1 0 5 6 1 0 8 5 3 8 6 4 0 7
 6 9 7 2 8 6 5 8 4 1 9 3 6 8 5 4 9 8 6 9 8 1 1 1 3 1 4 7 2 6 8 9 2 5 1 9 4
 7 6 4 2 2 0 4 8 7 1 7 7 1 3 6 7 9 9 0 3 8 5 6 6 0 6 1 1 7 8 9 2 9 8 7 6 1
 5 2 5 1 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 9 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 4 8 2 2 2 9 3 8 3 0 7 5 8 4 1 7 7 2 1 4 9 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 8 1 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 5 1 5 4 3 1 2 2 8 8 2 2 2 2 9 3 7 8 0
 0 1 1 1 4 8 5 4 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 0 1 6 8 9 0 2 6 8 3 1 4 1
 5 8 7 4 0 0 5 4 7 7 6 7 3 1 4 0 1 9 8 1 4 7 8 1 8 9 2]
Model Score of Kernal(rbf) : 0.9944444444444445
0.9944444444444445
```

In [46]:
```
## Polynomial kernel function
model4=SVC(kernel='poly',random_state=0, probability=True)

model4.fit(X_train,y_train)

y_pred_4=model4.predict(X_test)
print("Predicted classes of hand written digits")
print(y_pred_4)
print(" Original classes of hand written digits")
print(y_test)

print("Model Score of Kernal(poly) :", model4.score(X_test,y_test))
acc=accuracy_score(y_pred_4,y_test)
print(acc)
```

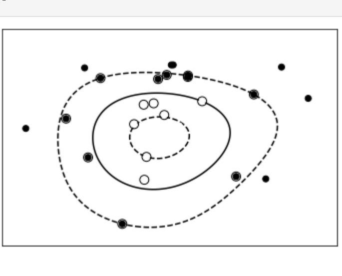
```
Predicted classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 8 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 8 0 4 2 7 6 1 0 5 6 1 0 8 9 3 8 6 4 0 7
 6 9 7 2 8 6 5 8 4 1 9 3 6 8 5 4 9 1 6 9 8 1 1 1 3 1 4 7 2 6 8 9 2 5 1 9 4
 7 6 4 2 2 0 4 8 7 1 7 7 1 3 6 7 9 9 0 3 8 5 6 6 0 6 1 1 7 8 9 2 9 8 7 6 1
 5 2 5 1 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 9 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 4 8 2 2 2 9 3 8 3 0 7 5 8 4 1 7 7 2 1 4 9 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 8 1 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 5 1 5 4 3 1 2 2 8 8 2 2 2 2 9 3 7 8 0
 0 1 1 1 4 8 5 4 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 0 1 6 5 9 0 2 6 8 3 1 4 1
 5 8 7 4 0 0 5 4 7 7 6 7 3 1 4 0 1 9 8 1 4 7 8 1 8 9 2]
 Original classes of hand written digits
[5 1 8 2 3 2 3 3 4 2 5 4 8 6 8 0 7 8 2 4 7 8 5 8 1 7 3 0 8 8 2 1 4 2 2 7 5
 4 9 4 9 4 0 1 6 5 6 0 7 5 8 7 2 6 8 0 4 2 7 6 1 0 5 6 1 0 8 5 3 8 6 4 0 7
 6 9 7 2 8 6 5 8 4 1 9 3 6 8 5 4 9 8 6 9 8 1 1 1 3 1 4 7 2 6 8 9 2 5 1 9 4
 7 6 4 2 2 0 4 8 7 1 7 7 1 3 6 7 9 9 0 3 8 5 6 6 0 6 1 1 7 8 9 2 9 8 7 6 1
 5 2 5 1 9 7 0 0 8 9 7 3 0 0 1 6 6 2 7 7 5 0 5 9 8 4 1 1 1 6 6 5 0 4 6 6 5
 8 7 6 9 5 2 9 2 2 9 8 0 4 3 4 8 2 2 2 9 3 8 3 0 7 5 8 4 1 7 7 2 1 4 9 3 6
 2 7 1 1 3 6 2 3 0 1 6 8 1 6 5 3 2 4 4 8 1 3 3 5 3 9 9 1 7 9 2 0 9 8 5 7 9
 7 8 1 3 8 7 3 5 3 7 2 4 7 5 4 3 6 4 5 1 5 4 3 1 2 2 8 8 2 2 2 2 9 3 7 8 0
 0 1 1 1 4 8 5 4 5 7 3 5 8 8 0 3 4 8 9 7 0 0 2 7 0 1 6 8 9 0 2 6 8 3 1 4 1
 5 8 7 4 0 0 5 4 7 7 6 7 3 1 4 0 1 9 8 1 4 7 8 1 8 9 2]
Model Score of Kernal(poly) : 0.9916666666666667
0.9916666666666667
```

In [11]:
```
import numpy as np
from matplotlib import pyplot as plt
from sklearn import svm

def linear_model(rseed=42, n_samples=30):
    " Generate data according to a linear model"
    np.random.seed(rseed)

    data = np.random.normal(0, 10, (n_samples, 2))
    data[:n_samples // 2] -= 15
    data[n_samples // 2:] += 15

    labels = np.ones(n_samples)
    labels[:n_samples // 2] = -1

    return data, labels


X, y = linear_model()
clf = svm.SVC(kernel='linear')
clf.fit(X, y)

plt.figure(figsize=(6, 4))
ax = plt.subplot(111, xticks=[], yticks=[])
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.bone)

ax.scatter(clf.support_vectors_[:, 0],
           clf.support_vectors_[:, 1],
           s=80, edgecolors="k", facecolors="none")

delta = 1
y_min, y_max = -50, 50
x_min, x_max = -50, 50
x = np.arange(x_min, x_max + delta, delta)
y = np.arange(y_min, y_max + delta, delta)
X1, X2 = np.meshgrid(x, y)
Z = clf.decision_function(np.c_[X1.ravel(), X2.ravel()])
Z = Z.reshape(X1.shape)

ax.contour(X1, X2, Z, [-1.0, 0.0, 1.0], colors='k',
           linestyles=['dashed', 'solid', 'dashed'])
```

Out[11]:
```
<matplotlib.contour.QuadContourSet at 0x1b3e5e213a0>
```



In [12]:
```
def nonlinear_model(rseed=42, n_samples=30):
    radius = 40 * np.random.random(n_samples)
    far_pts = radius > 20
    radius[far_pts] *= 1.2
    radius[~far_pts] *= 1.1

    theta = np.random.random(n_samples) * np.pi * 2

    data = np.empty((n_samples, 2))
    data[:, 0] = radius * np.cos(theta)
    data[:, 1] = radius * np.sin(theta)

    labels = np.ones(n_samples)
    labels[far_pts] = -1

    return data, labels


X, y = nonlinear_model()
clf = svm.SVC(kernel='rbf', gamma=0.001, coef0=0, degree=3)
clf.fit(X, y)

plt.figure(figsize=(6, 4))
ax = plt.subplot(1, 1, 1, xticks=[], yticks=[])
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.bone, zorder=2)

ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
           s=80, edgecolors="k", facecolors="none")

delta = 1
y_min, y_max = -50, 50
x_min, x_max = -50, 50
x = np.arange(x_min, x_max + delta, delta)
y = np.arange(y_min, y_max + delta, delta)
X1, X2 = np.meshgrid(x, y)
Z = clf.decision_function(np.c_[X1.ravel(), X2.ravel()])
Z = Z.reshape(X1.shape)

ax.contour(X1, X2, Z, [-1.0, 0.0, 1.0], colors='k',
           linestyles=['dashed', 'solid', 'dashed'], zorder=1)

plt.show()
```



Quantifying Neural Networks: Quality Metrics analysis

In [6]:
```
# SCIKIT Library
# Python script for confusion matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
predicted = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1] #Number of  0's=6  and Number of  1's=9,
actual    = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1]
#Number of  0's=8  and Number of  1's=7

#TP: It is an outcome where the model correctly predicts the positive class.
#TN: It is an outcome where the model correctly predicts the negative class.
#FP: It is an outcome where the model incorrectly predicts the positive class
#FN: It is an outcome where the model incorrectly predicts the negative  class
#TP(1)=6,FP(1)=3, TN(0)=5,FN(0)=1
results = confusion_matrix(actual, predicted)
accu_predicted=accuracy_score(actual, predicted)
precison=precision_score(actual, predicted, average=None)
recall=recall_score(actual, predicted,average=None)
F1=f1_score(actual, predicted,average=None)

# precison=precision_score(actual, predicted, average='binary')
# recall=recall_score(actual, predicted,average='binary')
# F1=f1_score(actual, predicted,average='binary')
report=classification_report(actual, predicted)
print("Confusion Matrix:")
print(results)
print("Predicted Accuarcy :")
print(accu_predicted)
print("Precison score :")
print(precison)
print("Recall score :")
print(recall)
print("F1 score :")
print(F1)
print("Accuracy Report :",report)

# Analytical method
#TP:  It is an outcome where the model correctly predicts the positive class.
#TN: It is an outcome where the model correctly predicts the negative class.
#FP: It is an outcome where the model incorrectly predicts the positive class
#FN: It is an outcome where the model incorrectly predicts the negative  class

#TP(1)=6,FP(1)=3, TN(0)=5,FN(0)=1
## Class 1:
#Precision(Pr): TP/(TP+FP)=6/(6+3)=0.6666 ; ## Recall(Re): TP/(TP+FN)=6/(6+1)=0.857; F1 score= 2*Pr*Re/(Pr+Re)=
# Accuracy : 6/7=85.71%
## Class 0:
#Precision: TN/(TN+FN)=5/(5+1)=0.8333 ; ## Recall: TN/(TN+FP)=5/(5+3)=0.625 ; F1 score= 2*Pr*Re/(Pr+Re)=0.714
# Accuracy : 5/8= 62.5%
#Overall or Average Accuracy:(TP+TN)/(TP+TN+FP+FN) = 11/15=0.73 (73%)
```

```
Confusion Matrix:
[[5 3]
 [1 6]]
Predicted Accuarcy :
0.7333333333333333
Precison score :
[0.83333333 0.66666667]
Recall score :
[0.625      0.85714286]
F1 score :
[0.71428571 0.75      ]
Accuracy Report :               precision    recall  f1-score   support

           0       0.83      0.62      0.71         8
           1       0.67      0.86      0.75         7

    accuracy                           0.73        15
   macro avg       0.75      0.74      0.73        15
weighted avg       0.76      0.73      0.73        15
```

Loss Functions for Neural Networks

Example 1: Mean Square Error (MSE) and Root Mean Square Error (RMSE) Loss functions

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

## True and predicted values
y_pred = np.array([12,18,19.5,18,9,23,24])  ## Predicted value by ANN model
y_true = np.array([11,20,19,17,10,24,23])  ## Target or actual Value

# y_true =[11,20,19,17,10,24,23]            ## Target or actual Value
# y_pred =[12,18,19.5,18,9,23,24]    ## Predicted value by ANN model

print("predicted is: ", y_pred)
print(" True  is: ", y_true)
# print("predicted is: " + str(["%.8f" % elem for elem in y_pred]))
# print(" True  is: " + str(["%.8f" % elem for elem in y_true]))


# # MSE loss function

# Define loss function
def mse_loss(y_pred, y_true):
    squared_error = (y_pred - y_true) ** 2
    sum_squared_error = np.sum(squared_error)
    loss = sum_squared_error / y_true.size
    rmse_loss = np.sqrt(loss)
    return loss,rmse_loss

## Calling MSE function
loss, rmse_loss = mse_loss(y_pred, y_true)
print("MSE error is: ", loss)
print("RMSE error is: ", rmse_loss)
# print(loss,rmse_loss)


# # Plotting
# x_vals = np.arange(-20, 20, 0.01)
# y_vals = np.square(x_vals)

# plt.plot(x_vals, y_vals, "blue")
# plt.grid(True, which="major")
# plt.show()
```

```
predicted is:  [12.  18.  19.5 18.   9.  23.  24. ]
 True  is:  [11 20 19 17 10 24 23]
MSE error is:  1.3214285714285714
RMSE error is:  1.14953406710222
```

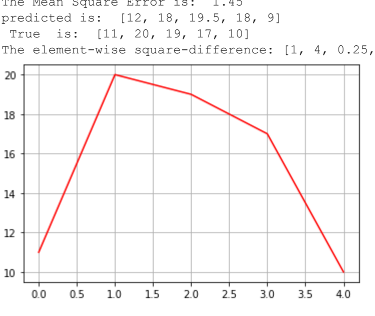
In [ ]:
```
#https://scikit-learn.org/stable/
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
```

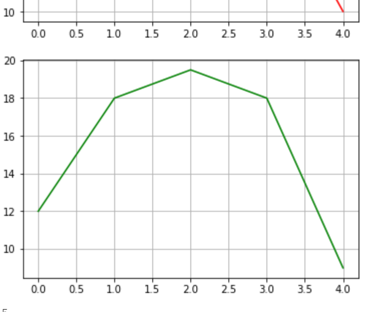
In [3]:
```
## Using SCIKIT Library

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
# y_pred = np.array([12,18,19.5,18,9,23,24])
# y_true = np.array([11,20,19,17,10,24,23])

y_true = [11,20,19,17,10,24,23]            ## Target or actual Value
y_pred = [12,18,19.5,18,9,23,24]      ## Predicted value by ANN model

print("predicted is: " + str(["%.8f" % elem for elem in y_pred]))
print(" True  is: " + str(["%.8f" % elem for elem in y_true]))

loss=mean_squared_error(y_true, y_pred)
rmse_loss=np.sqrt(loss)
print("MSE error is: ", loss)
print("RMSE error is: ", rmse_loss)
# print(loss,rmse_loss)


# # Plotting
# x_vals = np.arange(-20, 20, 0.01)
# y_vals = np.square(x_vals)

# plt.plot(x_vals, y_vals, "blue")
# plt.grid(True, which="major")
# plt.show()
```

```
predicted is: ['12.00000000', '18.00000000', '19.50000000', '18.00000000', '9.00000000', '23.00000000', '24.000
00000']
 True  is: ['11.00000000', '20.00000000', '19.00000000', '17.00000000', '10.00000000', '24.00000000', '23.00000
000']
MSE error is:  1.3214285714285714
RMSE error is:  1.14953406710222
```

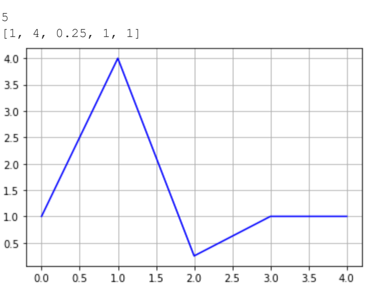
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
x_vals = np.arange(0,5,1)
y_true = [11, 20, 19, 17, 10]            ## Target or actual Value
y_pred = [12, 18, 19.5, 18, 9]      ## Predicted value by ANN model
summation = 0       #variable to store the summation of differences
n = len(y_true)          #finding total number of items in list
diff=[]
for i in range (n):  #looping through each element of the list
    difference = y_pred[i]- y_true[i]    #finding the difference between observed and predicted value
    # print(difference)
#       diff.append(difference)
    squared_difference = difference**2  #taking square of the differene
    diff.append(squared_difference)
    summation = summation + squared_difference  #taking a sum of all the differences
MSE = summation/n  #dividing summation by total values to obtain average
print("The Mean Square Error is: " , MSE)
print("predicted is: ", y_pred)
print(" True  is: ", y_true)
print("The element-wise square-difference:", diff)

## Plotting
plt.plot(x_vals,  y_true, "red")
plt.grid(True, which="major")
plt.show()
plt.plot(x_vals,  y_pred, "green")
plt.grid(True, which="major")
plt.show()
x_vals = np.arange(0,5,1)
n1 = len(x_vals)
print(n1)
# y_vals =np.abs(diff)
y_vals =diff
print(diff)
n2=len(y_vals)
plt.plot(x_vals, y_vals, "blue")
plt.grid(True, which="major")
plt.show()
```

```
The Mean Square Error is:  1.45
predicted is:  [12, 18, 19.5, 18, 9]
 True  is:  [11, 20, 19, 17, 10]
The element-wise square-difference: [1, 4, 0.25, 1, 1]
```





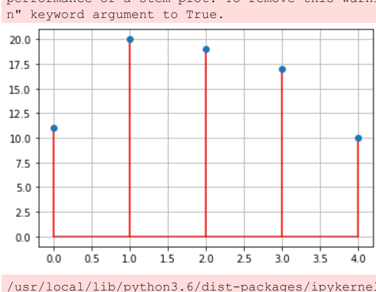

```
5
[1, 4, 0.25, 1, 1]
```



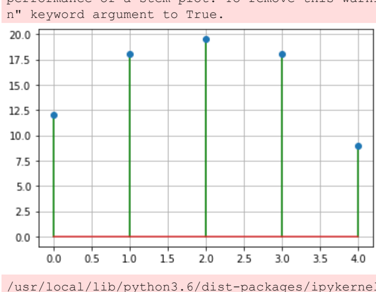
Example 2: Mean Absolute Error (MAE) Loss function

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
x_vals = np.arange(0,5,1)
# y_pred = [12,18,19.5,18,9]      ## Predicted value by ANN model
# y_true = [11,20,19,17,10]            ## Target or actual Value
y_pred = np.array([12,18,19.5,18,9])    ## Predicted value by ANN model
y_true = np.array([11,20,19,17,10])       ## Target or actual Value
summation = 0      #variable to store the summation of differences
n = len(y_true)    #finding total number of items in list
diff=[]
for i in range (n):  #looping through each element of the list
    difference = y_pred[i]-y_true[i]    #finding the difference between observed and predicted value
    abs_error = np.abs(difference)  #taking square of the differene
    diff.append(abs_error)
    summation = summation + abs_error #taking a sum of all the differences
MAE = summation/n  #dividing summation by total values to obtain average
print("The Mean Absolute Error is: " , MAE)
print("predicted is: ", y_pred)
print(" True  is: ", y_true)
print("The element-wise difference:", diff)

# Plotting the data
plt.stem(x_vals,  y_true, "red")
plt.grid(True, which="major")
plt.show()

plt.stem(x_vals,  y_pred, "green")
plt.grid(True, which="major")
plt.show()
# Plotting loss function
x_vals = np.arange(0,5,1)
n1 = len(x_vals)
print(n1)
# y_vals =np.abs(diff)
y_vals =diff
print(diff)
n2=len(y_vals)
plt.stem(x_vals, y_vals, "blue")
plt.grid(True, which="major")
plt.show()
```

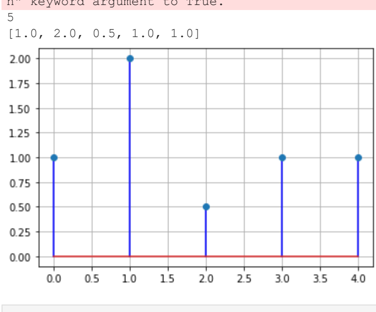
```
The Mean Absolute Error is:  1.1
predicted is:  [12.  18.  19.5 18.   9. ]
 True  is:  [11 20 19 17 10]
The element-wise difference: [1.0, 2.0, 0.5, 1.0, 1.0]
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:23: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
```



```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:27: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
```



```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:38: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use_line_collection" keyword argument to True.
5
[1.0, 2.0, 0.5, 1.0, 1.0]
```



In [ ]:

```
## Using SCIKIT Library
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error
y_pred = np.array([12,18,19.5,18,9])
y_true = np.array([11,20,19,17,10])

# y_true = [11,20,19,17,10,24,23]            ## Target or actual Value
# y_pred = [12,18,19.5,18,9,23,24]      ## Predicted value by ANN model
# y_true = [3, -0.5, 2, 7]
# y_pred = [2.5, 0.0, 2, 8]
print("predicted is: " + str(["%.8f" % elem for elem in y_pred]))
print(" True  is: " + str(["%.8f" % elem for elem in y_true]))

loss=mean_absolute_error(y_true, y_pred)
print("MSE error is: ", loss)
```

```
predicted is: ['12.00000000', '18.00000000', '19.50000000', '18.00000000', '9.00000000']
 True  is: ['11.00000000', '20.00000000', '19.00000000', '17.00000000', '10.00000000']
MSE error is:  1.1
```

Binary Cross ertropy Loss function

In [ ]:

```
#Probability that the element belongs to class 1 (or positive class) = p
#Then, the probability that the element belongs to class 0 (or negative class) = 1 - p
```

In [41]:

```
import numpy as np
from sklearn.metrics import log_loss

# y_true = np.array([1., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 1.])      #Number of  0's=8  and Nu
# y_pred = np.array([1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 0., 1., 1.])      #Number of  0's=6  and Nu
# targets     = np.array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1])      #Number of  0's=8  and Number of  1
# predictions = np.array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1])      #Number of  0's=8 and Number of  1
# y_pred =np.array([[0.1,0.1,0.98,0.1],[0.98,0.98,0.98,0.1],[0.98,0.98,0.1,0.98]])
# y_true =np.array([[1.,1.,0.,1.], [0.,0., 1.,0.],[0.,0.,1.,1.]])

y_true = np.array ([[0,0,1,0],[0,0,1,0],[0,0,1.,0.]])
y_pred = np.array ([[0.1,0.1,0.98,0.1],[0.1,0.98,0.1,0.1],[0.98, 0.1,0.98, 0.1]])
BCE=log_loss(y_true, y_pred)
mean_BCE_score = 1.0 / len(y_true) * BCE
print(mean_BCE_score)

# from tensorflow import keras
# from keras.losses import binary_crossentropy
# BCE=binary_crossentropy(y_true, y_pred)
# mean_BCE_score = 1.0 / len(y_true) * BCE
# print(mean_BCE_score)
```

```
0.4007576539098011
```

In [39]:

```
import numpy as np
from math import log
# calculate binary cross entropy
def binary_cross_entropy(actual, predicted):
    sum_score = 0.0
    for i in range(len(actual)):
        sum_score += actual[i] * log(1e-10 + predicted[i])         #Log is undefined for p=0 or p=1, so probabi
        mean_sum_score = 1.0 / len(actual) * sum_score
    return  -mean_sum_score

# Test results
# y_pred = [1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1]    #Number of  0's=6  and Number of  1's=9,
# y_true  = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1]    #Number of  0's=8  and Number of  1's=7
y_true =[0,0,1,0,0,0,1,0,0,0,1.,0]
y_pred = [0.1,0.1,0.98,0.1,0.1,0.98,0.1,0.1,0.98, 0.1,0.98, 0.1]
BCE=binary_cross_entropy(y_true, y_pred)
print(BCE)
```

```
0.19524920886875022
```

In [ ]: