Machine Learning

Indah Agustien Siradjuddin

Logistic Regression ¶

Semester Gasal 2019-2020

Regression Vs. Classification

Regression : input - real value, output - real value
Classification : input - real value, output - kelas

Binary Classifier

- map the input into output, which has only two kinds of value, i.e. positive or negative, 1 or 0
- Target data : only two values, i.e. 1 or 0

E.g. :

- Email : Spam or ham
- Cancer : benign or malignant
- CC transaction : Fraud or regular transaction

Output :

$y \in 0, 1$, where:

- 0 is negative class
- 1 is positive class

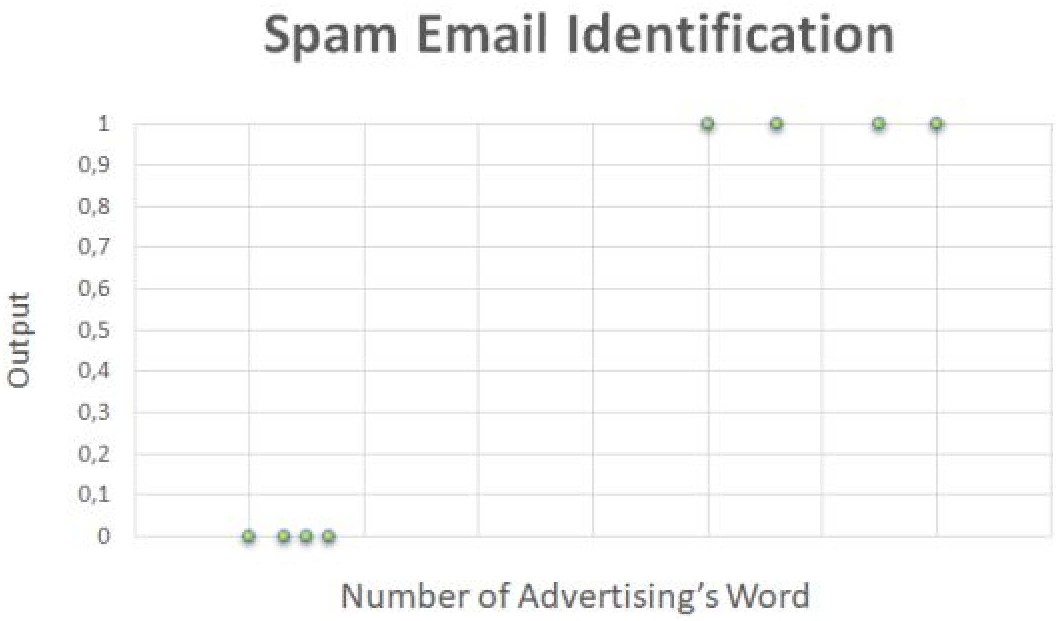
note: depends on how you determine which one is the positive or negative class

Linear Regression for Classification

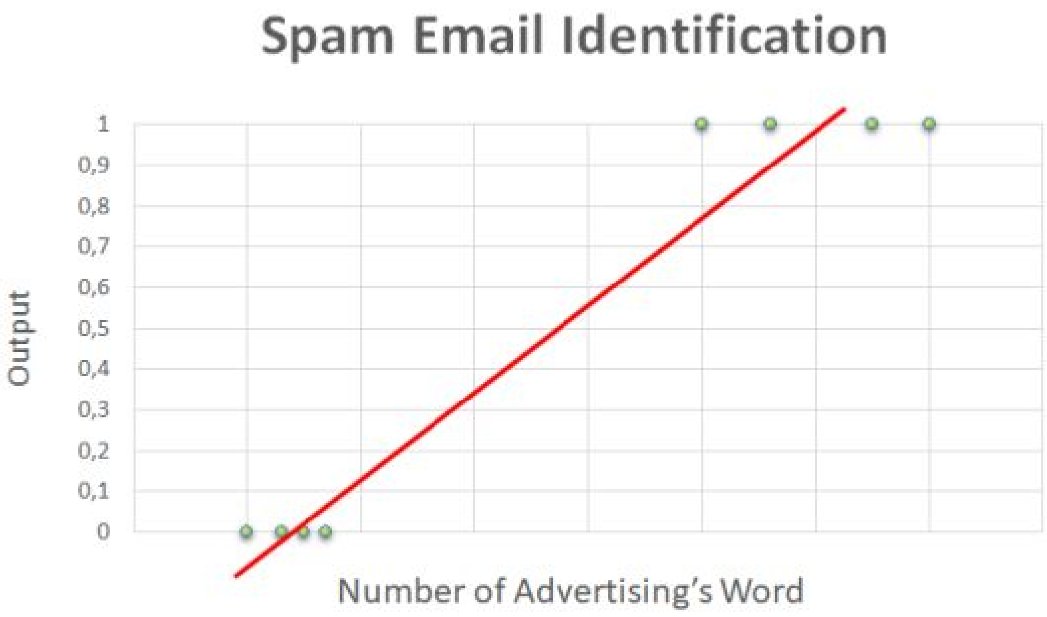
Spam Email Identification

based on number of advertising's words.
Output value :

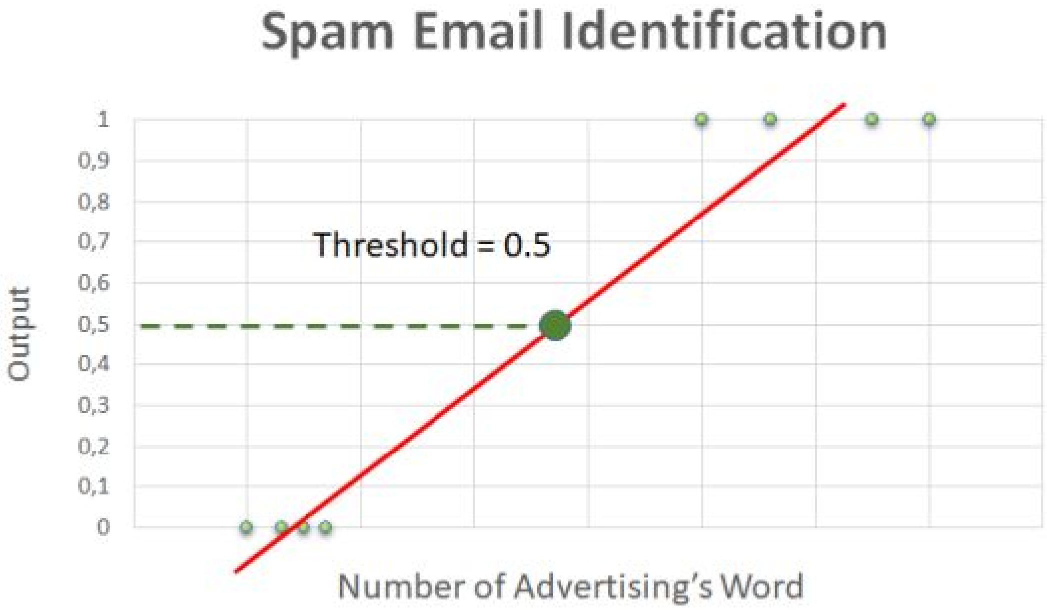
- Spam : 1
- Ham : 0



based on the training dataset, below is the trained model

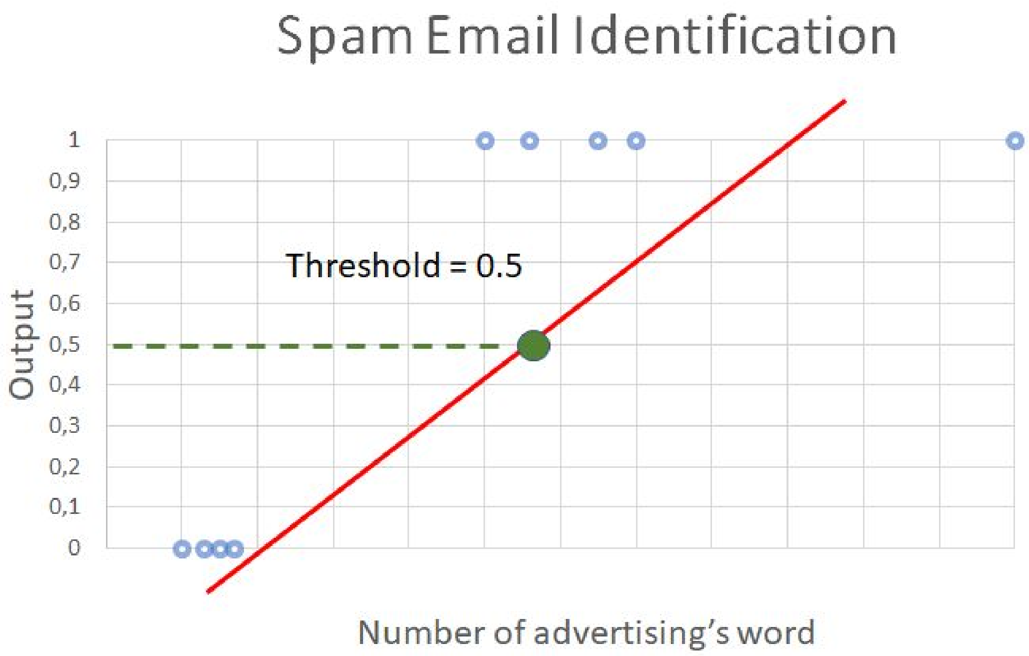


Linear Regression for classification --> using threshold Threshold is used to classify the data: $output < threshold$ then $output = 0$, else $output = 1$



based on the threshold, which data are belongs to spam or ham ?

However, if new data is included, then the model should be updated



based on the threshold, which data are belongs to spam or ham ?

Therefore, Linear Regression Model can't be used for the classification

Logistic Regression

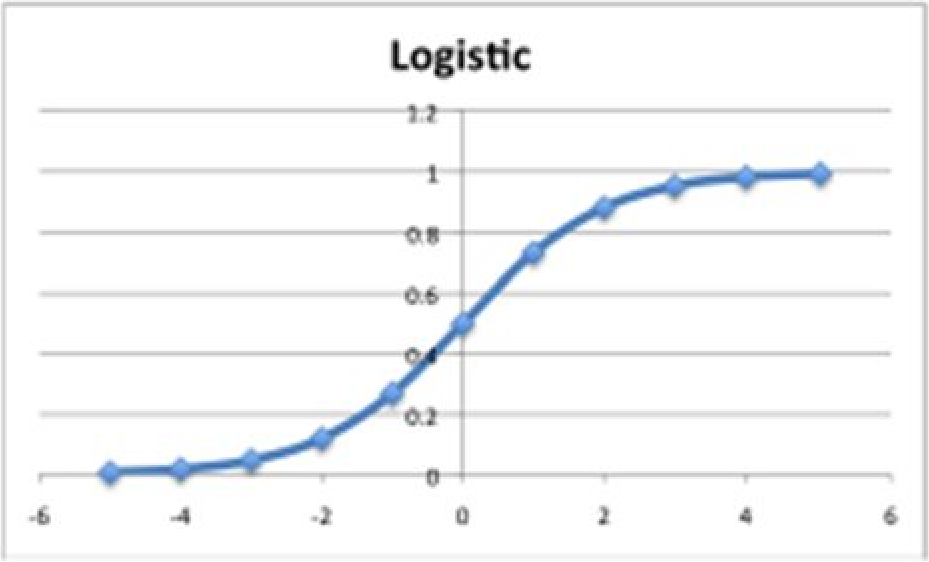
Definition

- predict the probability of a data belongs to certain class
- binary classifier
- Supervised classifier

Logistic Function

output of logistic function falls to $0 \leq y \leq 1$ Fungsi logistik :

$$y = f(x) = \frac{1}{1 + e^{-x}}$$



Linear Regression :

$$f(x; w) = w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d$$

Logistic Regression :

$$f(x; w) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2 + \ldots + w_dx_d)}}$$

Update weight with Gradient descent :

$$w_j = w_j + \eta(y^i - f(x^i))f(x^i)(1 - f(x^i))x_j^i$$

In [3]:

```
x=np.array([[3,3],[1,2],[3,4],[1,2],[3,3],[8,3],[5,2],[7,2],[9,0],[8,4]])
target=np.array([[0],[0],[0],[0],[0],[1],[1],[1],[1],[1]])
test=np.array([[1,2]])
```

Logistic Regression with Scikit

Training/Learning

- load dataset (features and target)
- create Model
- Training/Learning

Testing

- Load Model
- test model with new Data / Generalize

In [2]:

```
from sklearn.linear_model.logistic import LogisticRegression
import numpy as np
```

from three lines code, define :

- data training (input) ? x
- target ? target
- number of data training ? 10
- number of attribute ? 2
- data test ? test

In [4]:

```
classifier = LogisticRegression()
classifier.fit(x,target)
print(classifier.intercept_,classifier.coef_)
pred=classifier.predict(test)
print('prediction = ',pred)
```

```
[-0.40885175] [[ 0.83023895 -1.10760329]]
prediction =  [0]

C:\Users\Indah Agustin\Anaconda3\lib\site-packages\sklearn\linear_model\lo
gistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in
0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\Indah Agustin\Anaconda3\lib\site-packages\sklearn\utils\validatio
n.py:724: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for examp
le using ravel().
  y = column_or_1d(y, warn=True)
```

Logistic Regression with your own code

In [5]:

```
import simpleLinear
```

In [6]:

```
simpleLinear.logRegression(10,0.3,x,target)
```

```
w= [[0.]
 [0.]
 [0.]]
output= [0.5]  error - 0 = [-0.5]
w= [[-0.0375]
 [-0.1125]
 [-0.1125]]
output= [0.4073334]  error - 1 = [-0.4073334]
w= [[-0.06700066]
 [-0.14200066]
 [-0.17150132]]
output= [0.23523062]  error - 2 = [-0.23523062]
w= [[-0.07969586]
 [-0.18008625]
 [-0.22228211]]
output= [0.3308493]  error - 3 = [-0.3308493]
w= [[-0.10166968]
 [-0.20206008]
 [-0.26622976]]
output= [0.18145225]  error - 4 = [-0.18145225]
w= [[-0.10975487]
 [-0.22631563]
 [-0.29048531]]
output= [0.05777239]  error - 5 = [0.94222761]
w= [[-0.09436789]
 [-0.10321983]
 [-0.24432439]]
output= [0.2499056]  error - 6 = [0.7500944]
w= [[-0.05218571]
 [ 0.1076911 ]
 [-0.15996002]]
output= [0.59429076]  error - 7 = [0.40570924]
w= [[-0.02283963]
 [ 0.31311363]
 [-0.10126787]]
output= [0.94241497]  error - 8 = [0.05758503]
w= [[-0.02190211]
 [ 0.32155135]
 [-0.10126787]]
output= [0.89524163]  error - 9 = [0.10475837]
w= [[-0.01895471]
 [ 0.34513054]
 [-0.08947827]]
output= [0.67874321]  error - 0 = [-0.67874321]
w= [[-0.06335487]
 [ 0.21193005]
 [-0.22267876]]
output= [0.42634426]  error - 1 = [-0.42634426]
w= [[-0.09463679]
 [ 0.18064813]
 [-0.2852426 ]]
output= [0.33321876]  error - 2 = [-0.33321876]
w= [[-0.11684756]
 [ 0.11401584]
 [-0.37408566]]
output= [0.32060278]  error - 3 = [-0.32060278]
w= [[-0.13779734]
 [ 0.09306605]
 [-0.41598523]]
output= [0.24851372]  error - 4 = [-0.24851372]
w= [[-0.15172067]
```

```
 [ 0.05129607]
 [-0.45775522]]
output= [0.24701099]  error - 5 = [0.75298901]
w= [[-0.10970466]
 [ 0.38742415]
 [-0.33170719]]
output= [0.76205908]  error - 6 = [0.23794092]
w= [[-0.09676127]
 [ 0.45214111]
 [-0.3058204 ]]
output= [0.92104172]  error - 7 = [0.07895828]
w= [[-0.09503862]
 [ 0.46419963]
 [-0.30237511]]
output= [0.98341868]  error - 8 = [0.01658132]
w= [[-0.09495751]
 [ 0.46492966]
 [-0.30237511]]
output= [0.91796245]  error - 9 = [0.08203755]
w= [[-0.0931041 ]
 [ 0.47975695]
 [-0.29496147]]
output= [0.61331834]  error - 0 = [-0.61331834]
w= [[-0.13674028]
 [ 0.3488484 ]
 [-0.42587001]]
output= [0.34532975]  error - 1 = [-0.34532975]
w= [[-0.16016162]
 [ 0.32542706]
 [-0.4727127 ]]
output= [0.25450437]  error - 2 = [-0.25450437]
w= [[-0.1746479 ]
 [ 0.28196822]
 [-0.53065782]]
output= [0.27808204]  error - 3 = [-0.27808204]
w= [[-0.1913956 ]
 [ 0.26522053]
 [-0.5641532 ]]
output= [0.25195858]  error - 4 = [-0.25195858]
w= [[-0.205642  ]
 [ 0.22248132]
 [-0.60689241]]
output= [0.43869287]  error - 5 = [0.56130713]
w= [[-0.16417688]
 [ 0.5542023 ]
 [-0.48249705]]
output= [0.83778522]  error - 6 = [0.16221478]
w= [[-0.15756332]
 [ 0.58727006]
 [-0.46926994]]
output= [0.95323771]  error - 7 = [0.04676229]
w= [[-0.15693799]
 [ 0.59164742]
 [-0.46801927]]
output= [0.99433568]  error - 8 = [0.00566432]
w= [[-0.15692842]
 [ 0.59173355]
 [-0.46801927]]
output= [0.93731299]  error - 9 = [0.06268701]
w= [[-0.15582342]
 [ 0.60057353]
 [-0.46359928]]
output= [0.56343122]  error - 0 = [-0.56343122]
w= [[-0.19740067]
 [ 0.47584178]
 [-0.58833103]]
output= [0.28941623]  error - 1 = [-0.28941623]
w= [[-0.21525659]
 [ 0.45798586]
 [-0.62404287]]
output= [0.20792634]  error - 2 = [-0.20792634]
w= [[-0.22552979]
 [ 0.42716625]
 [-0.66513568]]
output= [0.24441311]  error - 3 = [-0.24441311]
w= [[-0.23907092]
 [ 0.41362513]
 [-0.69221793]]
output= [0.25448197]  error - 4 = [-0.25448197]
w= [[-0.25355508]
 [ 0.37017264]
 [-0.73567042]]
output= [0.62265078]  error - 5 = [0.37734922]
w= [[-0.22695685]
 [ 0.58295846]
 [-0.65587574]]
output= [0.79836133]  error - 6 = [0.20163867]
w= [[-0.21721888]
 [ 0.6316483 ]
 [-0.6363998 ]]
output= [0.94938275]  error - 7 = [0.05061725]
w= [[-0.21648916]
 [ 0.63675638]
 [-0.63494035]]
output= [0.99598749]  error - 8 = [0.00401251]
w= [[-0.21648435]
 [ 0.63679968]
 [-0.63494035]]
output= [0.91198784]  error - 9 = [0.08801216]
w= [[-0.21436503]
 [ 0.65375421]
 [-0.62646309]]
output= [0.46692545]  error - 0 = [-0.46692545]
w= [[-0.24923121]
 [ 0.54915568]
 [-0.73106161]]
output= [0.23826799]  error - 1 = [-0.23826799]
w= [[-0.26220464]
 [ 0.53618225]
 [-0.75700847]]
output= [0.15687158]  error - 2 = [-0.15687158]
w= [[-0.26842912]
 [ 0.51750879]
 [-0.78190642]]
output= [0.21169589]  error - 3 = [-0.21169589]
w= [[-0.27902751]
 [ 0.5069104 ]
 [-0.8031032 ]]
output= [0.237288]  error - 4 = [-0.237288]
w= [[-0.291911  ]
 [ 0.46825994]
 [-0.84175366]]
```

```
output= [0.7168536]  error - 5 = [0.2831464]
w= [[-0.27466955]
 [ 0.60619155]
 [-0.7900293 ]]
output= [0.7642692]  error - 6 = [0.2357308]
w= [[-0.26192864]
 [ 0.66989607]
 [-0.76454749]]
output= [0.94775981]  error - 7 = [0.05224019]
w= [[-0.2611527 ]
 [ 0.67532767]
 [-0.76299561]]
output= [0.99703171]  error - 8 = [0.00296829]
w= [[-0.26115007]
 [ 0.67535139]
 [-0.76299561]]
output= [0.88989594]  error - 9 = [0.11010406]
w= [[-0.25791363]
 [ 0.70124288]
 [-0.75004986]]
output= [0.40027136]  error - 0 = [-0.40027136]
w= [[-0.28673968]
 [ 0.61476474]
 [-0.836528  ]]
output= [0.20668394]  error - 1 = [-0.20668394]
w= [[-0.2969064 ]
 [ 0.60459802]
 [-0.85686144]]
output= [0.12891816]  error - 2 = [-0.12891816]
w= [[-0.30124959]
 [ 0.59156846]
 [-0.87423419]]
output= [0.18875052]  error - 3 = [-0.18875052]
w= [[-0.30992024]
 [ 0.58289781]
 [-0.8915755 ]]
output= [0.22514115]  error - 4 = [-0.22514115]
w= [[-0.32170318]
 [ 0.54754899]
 [-0.92692432]]
output= [0.78209479]  error - 5 = [0.21790521]
w= [[-0.31056239]
 [ 0.63667529]
 [-0.89350196]]
output= [0.74759192]  error - 6 = [0.25240808]
w= [[-0.29627371]
 [ 0.70811873]
 [-0.86492458]]
output= [0.94934375]  error - 7 = [0.05065625]
w= [[-0.29554288]
 [ 0.71323447]
 [-0.86346294]]
output= [0.99781416]  error - 8 = [0.00218584]
w= [[-0.29554145]
 [ 0.71324735]
 [-0.86346294]]
output= [0.87616296]  error - 9 = [0.12383704]
w= [[-0.29151051]
 [ 0.74549494]
 [-0.84733914]]
output= [0.35502047]  error - 0 = [-0.35502047]
```

```
w= [[-0.31589838]
 [ 0.67233131]
 [-0.92050277]]
output= [0.18473775]  error - 1 = [-0.18473775]
w= [[-0.32424537]
 [ 0.66398432]
 [-0.93719675]]
output= [0.11094945]  error - 2 = [-0.11094945]
w= [[-0.32752858]
 [ 0.6541347 ]
 [-0.95032957]]
output= [0.17163937]  error - 3 = [-0.17163937]
w= [[-0.33484965]
 [ 0.64681364]
 [-0.96497171]]
output= [0.21596727]  error - 4 = [-0.21596727]
w= [[-0.34582027]
 [ 0.61390176]
 [-0.99788358]]
output= [0.82803196]  error - 5 = [0.17196804]
w= [[-0.33847405]
 [ 0.67267151]
 [-0.97584493]]
output= [0.74520379]  error - 6 = [0.25479621]
w= [[-0.32396021]
 [ 0.7452407 ]
 [-0.94681725]]
output= [0.95253301]  error - 7 = [0.04746699]
w= [[-0.32331636]
 [ 0.74974765]
 [-0.94552955]]
output= [0.99838114]  error - 8 = [0.00161886]
w= [[-0.32331558]
 [ 0.74975471]
 [-0.94552955]]
output= [0.86905214]  error - 9 = [0.13094786]
w= [[-0.318845  ]
 [ 0.78551935]
 [-0.92764723]]
output= [0.32186185]  error - 0 = [-0.32186185]
w= [[-0.33992053]
 [ 0.72229278]
 [-0.99087381]]
output= [0.16806893]  error - 1 = [-0.16806893]
w= [[-0.34697043]
 [ 0.71524287]
 [-1.00497363]]
output= [0.09786843]  error - 2 = [-0.09786843]
w= [[-0.34956268]
 [ 0.70746613]
 [-1.01534262]]
output= [0.15805365]  error - 3 = [-0.15805365]
w= [[-0.35587247]
 [ 0.70115634]
 [-1.02796219]]
output= [0.20812082]  error - 4 = [-0.20812082]
w= [[-0.36616237]
 [ 0.67028663]
 [-1.0588319 ]]
output= [0.86052232]  error - 5 = [0.13947768]
w= [[-0.36114018]
```

```
 [ 0.71046412]
 [-1.04376534]]
output= [0.75094333]  error - 6 = [0.24905667]
w= [[-0.34716605]
 [ 0.78033477]
 [-1.01581708]]
output= [0.95620879]  error - 7 = [0.04379121]
w= [[-0.34661595]
 [ 0.78418552]
 [-1.01471686]]
output= [0.9987841]  error - 8 = [0.0012159]
w= [[-0.3466155 ]
 [ 0.78418951]
 [-1.01471686]]
output= [0.86623053]  error - 9 = [0.13376947]
w= [[-0.34196533]
 [ 0.82139086]
 [-0.99611619]]
output= [0.29605785]  error - 0 = [-0.29605785]
w= [[-0.36047555]
 [ 0.76586023]
 [-1.05164682]]
output= [0.15473856]  error - 1 = [-0.15473856]
w= [[-0.36654723]
 [ 0.75978854]
 [-1.0637902 ]]
output= [0.08767637]  error - 2 = [-0.08767637]
w= [[-0.36865118]
 [ 0.75347669]
 [-1.072206  ]]
output= [0.14684214]  error - 3 = [-0.14684214]
w= [[-0.37417008]
 [ 0.7479578 ]
 [-1.08324378]]
output= [0.2010045]  error - 4 = [-0.2010045]
w= [[-0.38385458]
 [ 0.7189043 ]
 [-1.11229728]]
output= [0.88396114]  error - 5 = [0.11603886]
w= [[-0.38028381]
 [ 0.74747043]
 [-1.10158498]]
output= [0.76022225]  error - 6 = [0.23977775]
w= [[-0.36717149]
 [ 0.81303204]
 [-1.07536034]]
output= [0.95981829]  error - 7 = [0.04018171]
w= [[-0.36670658]
 [ 0.81628639]
 [-1.07443052]]
output= [0.99907044]  error - 8 = [0.00092956]
w= [[-0.36670632]
 [ 0.81628872]
 [-1.07443052]]
output= [0.865981]  error - 9 = [0.134019]
w= [[-0.36204013]
 [ 0.85361824]
 [-1.05576577]]
output= [0.27518303]  error - 0 = [-0.27518303]
w= [[-0.37850631]
 [ 0.80421969]
 [-1.10516431]]
output= [0.14373418]  error - 1 = [-0.14373418]
w= [[-0.38381332]
 [ 0.79891268]
 [-1.11577833]]
output= [0.07942476]  error - 2 = [-0.07942476]
w= [[-0.3855555 ]
 [ 0.79368615]
 [-1.12274704]]
output= [0.13736341]  error - 3 = [-0.13736341]
w= [[-0.39043855]
 [ 0.7888031 ]
 [-1.13251314]]
output= [0.19441578]  error - 4 = [-0.19441578]
w= [[-0.39957327]
 [ 0.76139894]
 [-1.1599173 ]]
output= [0.90131006]  error - 5 = [0.09868994]
w= [[-0.39693973]
 [ 0.78246732]
 [-1.15201666]]
output= [0.77054013]  error - 6 = [0.22945987]
w= [[-0.38476862]
 [ 0.84332285]
 [-1.12767445]]
output= [0.96314251]  error - 7 = [0.03685749]
w= [[-0.3843761 ]
 [ 0.8460705 ]
 [-1.12688941]]
output= [0.99927619]  error - 8 = [0.00072381]
w= [[-0.38437594]
 [ 0.84607191]
 [-1.12688941]]
output= [0.86722491]  error - 9 = [0.13277509]
```

Accuracy

The simplest performance measure for classification is calculate the true prediction of all data test :

$$akurasi = \frac{True}{NumberOfData}$$

In [7]:

```python
from sklearn.metrics import accuracy_score
import numpy as np
```

In [8]:

```python
target=np.array([0,0,0,0,0,1,1,1,1,1])
pred=np.array([0,1,0,0,0,0,0,1,1,1])
acc=accuracy_score(target,pred)
print(acc)
```

```
0.7
```

Confusion Matrix

Calculate True Positive, True Negative, False Positive, and False Negative

Positive - Negative : class
True - False : is predicted result equal to the target class
Email identification : spam identification
Positive class : Spam Email Negative class : ham email

True Positive : if data is predicted into positive class, and the target is positive class, i.e. spam email (output = spam, target =spam

True Negative : if data is predicted into negative class, and the target is negative class, i.e. ham email (output = ham, target = ham)

False Positive : if data is predicted into positive class, and the target is negative class, (output = spam, target = ham)

False Negative : if data is predicted into negative class, and the target is positive class, (output = ham, target = spam)

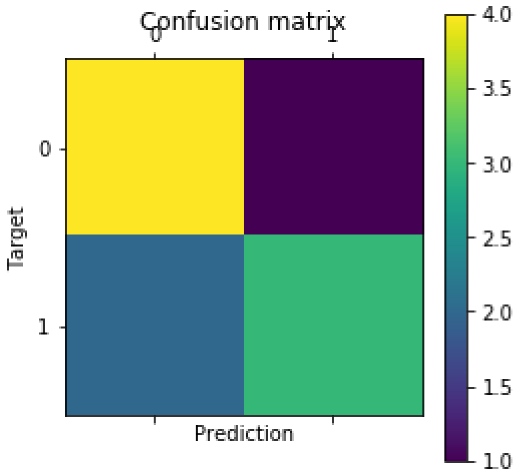
In [9]:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

In [10]:

```
target=np.array([0,0,0,0,0,1,1,1,1,1])
pred=np.array([0,1,0,0,0,0,0,1,1,1])
confusionMatrix = confusion_matrix(target, pred)
print(confusionMatrix)
plt.matshow(confusionMatrix)
plt.title('Confusion matrix')
plt.colorbar()
plt.ylabel('Target')
plt.xlabel('Prediction')
plt.show()
```

```
[[4 1]
 [2 3]]
```



Precision - Recall

$precision = \frac{TP}{TP+FP}$

$recall = \frac{TP}{TP+FN}$

In [11]:

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
print(precision_score(target, pred))
print(recall_score(target, pred))
```

```
0.75
0.6
```