Assignment-16- Neural-Network-02-(Gas_Turbines)

In [1]:

```
# Importing libraries
import pandas as pd
import numpy as np
import keras
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder,LabelEncoder
```

In [2]:

```
#Loading dataset
gas_turbines=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/gas_turbines.cs
```

In [3]:

```
gas_turbines.head()
```

Out[3]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	114.70	10.605	3.1547	82.722
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	114.72	10.598	3.2363	82.776
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	114.71	10.601	3.2012	82.468
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	114.72	10.606	3.1923	82.670
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	114.72	10.612	3.2484	82.311

In [4]:

```
# describing the dataset

gas_turbines.describe()
```

Out[4]:

	AT	AP	AH	AFDP	GTEP	TIT	
count	15039.000000	15039.00000	15039.000000	15039.000000	15039.000000	15039.000000	150
mean	17.764381	1013.19924	79.124174	4.200294	25.419061	1083.798770	5
std	7.574323	6.41076	13.793439	0.760197	4.173916	16.527806	
min	0.522300	985.85000	30.344000	2.087400	17.878000	1000.800000	5
25%	11.408000	1008.90000	69.750000	3.723900	23.294000	1079.600000	5
50%	18.186000	1012.80000	82.266000	4.186200	25.082000	1088.700000	5
75%	23.862500	1016.90000	90.043500	4.550900	27.184000	1096.000000	5
max	34.929000	1034.20000	100.200000	7.610600	37.402000	1100.800000	5

In [5]:

```
# lets explore about the data types, null cols, dataset length, rows and columns

gas_turbines.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15039 entries, 0 to 15038
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AT      15039 non-null  float64
 1   AP      15039 non-null  float64
 2   AH      15039 non-null  float64
 3   AFDP    15039 non-null  float64
 4   GTEP    15039 non-null  float64
 5   TIT     15039 non-null  float64
 6   TAT     15039 non-null  float64
 7   TEY     15039 non-null  float64
 8   CDP     15039 non-null  float64
 9   CO      15039 non-null  float64
 10  NOX     15039 non-null  float64
dtypes: float64(11)
memory usage: 1.3 MB
```

In [6]:

```
gas_turbines.shape
```

Out[6]:

```
(15039, 11)
```

In [7]:

```
# finding NA values null values

gas_turbines.isna().sum()
```

Out[7]:

```
AT      0
AP      0
AH      0
AFDP    0
GTEP    0
TIT     0
TAT     0
TEY     0
CDP     0
CO      0
NOX     0
dtype: int64
```

No. Na \ null values present the dataset

In [8]:

```
# making the dataset safe

gas_turbo = gas_turbines.copy()
```

In [9]:

```
gas_turbo.head()
```

Out[9]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	TEY	CDP	CO	NOX
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	114.70	10.605	3.1547	82.722
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	114.72	10.598	3.2363	82.776
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	114.71	10.601	3.2012	82.468
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	114.72	10.606	3.1923	82.670
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	114.72	10.612	3.2484	82.311

In [10]:

```
gas_turbo.TEY.mean()
```

Out[10]:

```
134.18846399361655
```

Data cleaning

Creating the TEY cols into categorical form.

In [11]:

```
gas_turbo.loc[gas_turbo['TEY']<=134,'TEY_New']='Low'
gas_turbo.loc[gas_turbo['TEY']>=134,'TEY_New']='High'
```

In [12]:

```
gas_turbo.drop('TEY',inplace = True,axis = 1)
```

In [13]:

```
gas_turbo.head()
```

Out[13]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	CDP	CO	NOX	TEY_New
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	10.605	3.1547	82.722	Low
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	10.598	3.2363	82.776	Low
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	10.601	3.2012	82.468	Low
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	10.606	3.1923	82.670	Low
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	10.612	3.2484	82.311	Low

Making the target cols dummies

In [17]:

```
#LebelEncoder - Encode target labels with value between 0 and n_classes-1.
# This transformer should be used to encode target values, i.e. y, and not the input X.

encoder = LabelEncoder()
target_encoded = encoder.fit_transform(gas_turbo['TEY_New'])
gas_turbo['TEY_New'] = target_encoded
encoder.inverse_transform(target_encoded)
```

Out[17]:

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

In [18]:

```
gas_turbo.head()
```

Out[18]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	CDP	CO	NOX	TEY_New
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	10.605	3.1547	82.722	1
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	10.598	3.2363	82.776	1
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	10.601	3.2012	82.468	1
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	10.606	3.1923	82.670	1
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	10.612	3.2484	82.311	1

Dividing the data into independent and dependent var.

In [19]:

```
X = gas_turbo.iloc[:,:-1]
y = gas_turbo['TEY_New']
```

In [20]:

```
X.head()
```

Out[20]:

	AT	AP	AH	AFDP	GTEP	TIT	TAT	CDP	CO	NOX
0	6.8594	1007.9	96.799	3.5000	19.663	1059.2	550.00	10.605	3.1547	82.722
1	6.7850	1008.4	97.118	3.4998	19.728	1059.3	550.00	10.598	3.2363	82.776
2	6.8977	1008.8	95.939	3.4824	19.779	1059.4	549.87	10.601	3.2012	82.468
3	7.0569	1009.2	95.249	3.4805	19.792	1059.6	549.99	10.606	3.1923	82.670
4	7.3978	1009.7	95.150	3.4976	19.765	1059.7	549.98	10.612	3.2484	82.311

In [21]:

```
y.head()
```

Out[21]:

```
0    1
1    1
2    1
3    1
4    1
Name: TEY_New, dtype: int64
```

In [22]:

```
# Standardization can be helpful in cases where the data follows a Gaussian distribution.
#However, this does not have to be necessarily true. Also, unlike normalization,
#standardization does not have a bounding range. So, even if you have outliers in your data
#they will not be affected by standardization.

# Standardization

a = StandardScaler()
a.fit(X)
X_standardized = a.transform(X)
```

In [23]:

```
pd.DataFrame(X_standardized).describe()
```

Out[23]:

	0	1	2	3	4	
count	1.503900e+04	1.503900e+04	1.503900e+04	1.503900e+04	1.503900e+04	1.503900e+0
mean	-2.320107e-16	-1.925280e-14	1.844983e-16	3.810001e-16	1.107344e-16	-2.324212e-1
std	1.000033e+00	1.000033e+00	1.000033e+00	1.000033e+00	1.000033e+00	1.000033e+0
min	-2.276462e+00	-4.266288e+00	-3.536594e+00	-2.779497e+00	-1.806771e+00	-5.021933e+0
25%	-8.392292e-01	-6.706510e-01	-6.796337e-01	-6.266930e-01	-5.091458e-01	-2.540512e-0
50%	5.566605e-02	-6.227861e-02	2.277844e-01	-1.854065e-02	-8.075681e-02	2.965544e-0
75%	8.051309e-01	5.772924e-01	7.916582e-01	4.612196e-01	4.228638e-01	7.382490e-0
max	2.266234e+00	3.275970e+00	1.528011e+00	4.486233e+00	2.871006e+00	1.028678e+0

Turning of Hyperparameters :- Batch Size and Epoch

In [24]:

```
# Importing the necessary packages
from sklearn.model_selection import GridSearchCV, KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.optimizers import Adam
```

In [25]:

```
# create model
def create_model():
    model = Sequential()
    model.add(Dense(14, input_dim =10, kernel_initializer ='uniform', activation='relu'))
    model.add(Dense(8, kernel_initializer ='uniform', activation='relu'))
    model.add(Dense(1, kernel_initializer ='uniform', activation='sigmoid'))

    adam=Adam(lr=0.01)
    model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
    return model
```

In [26]:

```
# Create the model
model = KerasClassifier(build_fn = create_model,verbose = 0)
# Define the grid search parameters
batch_size = [10,20,40]
epochs = [10,50,100]
# Make a dictionary of the grid search parameters
param_grid = dict(batch_size = batch_size,epochs = epochs)
# Build and fit the GridSearchCV
grid = GridSearchCV(estimator = model,param_grid = param_grid,cv = KFold(),verbose = 10)
grid_result = grid.fit(X_standardized,y)
```

```
0.................................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 3/5; 6/9] END .................batch_size=20, epochs=100; total time=
1.3min
[CV 4/5; 6/9] START batch_size=20, epochs=10
0.................................



C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).  init  (name, **kwargs)
```

In [27]:

```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.917548394203186, using {'batch_size': 40, 'epochs': 10}
0.9056466341018676,0.04203890563699189 with: {'batch_size': 10, 'epochs': 1
0}
0.9057133078575135,0.03906312112012391 with: {'batch_size': 10, 'epochs': 5
0}
0.9120950102806091,0.02796467091587771 with: {'batch_size': 10, 'epochs': 10
0}
0.9085044622421264,0.036343511778729604 with: {'batch_size': 20, 'epochs': 1
0}
0.8991962313652039,0.04305163482162257 with: {'batch_size': 20, 'epochs': 5
0}
0.9038509368896485,0.03165470454823999 with: {'batch_size': 20, 'epochs': 10
0}
0.917548394203186,0.027379384182439136 with: {'batch_size': 40, 'epochs': 1
0}
0.9097696423530579,0.03570585347113867 with: {'batch_size': 40, 'epochs': 5
0}
0.9142233610153199,0.03266564708665834 with: {'batch_size': 40, 'epochs': 10
0}
```

From the above Hyperparameters the Batch size :- 40 and the epoch is :- 10

Tuning of Hyperparameters :- Learning rate and Drop out rate

In [28]:

```
from keras.layers import Dropout

# Defining the model

def create_model(learning_rate,dropout_rate):
    model = Sequential()
    model.add(Dense(14,input_dim = 10,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(8,input_dim = 14,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = learning_rate)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 40,epochs = 10)

# Define the grid search parameters

learning_rate = [0.001,0.01,0.1]
dropout_rate = [0.0,0.1,0.2]

# Make a dictionary of the grid search parameters

param_grids = dict(learning_rate = learning_rate,dropout_rate = dropout_rate)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 10)
grid_result = grid.fit(X_standardized,y)
```

```
  super(Adam, self).__init__(name, **kwargs)

[CV 2/5; 6/9] END .......dropout_rate=0.1, learning_rate=0.1; total time=
5.6s
[CV 3/5; 6/9] START dropout_rate=0.1, learning_rate=0.
1.........................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 3/5; 6/9] END .......dropout_rate=0.1, learning_rate=0.1; total time=
5.4s
[CV 4/5; 6/9] START dropout_rate=0.1, learning_rate=0.
1.........................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)
```

In [29]:

```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9144896626472473, using {'dropout_rate': 0.0, 'learning_rate': 0.00
1}
0.9144896626472473,0.024090924920750108 with: {'dropout_rate': 0.0, 'learnin
g_rate': 0.001}
0.8984638929367066,0.04748195087423267 with: {'dropout_rate': 0.0, 'learning
_rate': 0.01}
0.9077065348625183,0.028771470860872013 with: {'dropout_rate': 0.0, 'learnin
g_rate': 0.1}
0.9130268931388855,0.03411621578574495 with: {'dropout_rate': 0.1, 'learning
_rate': 0.001}
0.8970681190490722,0.0430271250699651 with: {'dropout_rate': 0.1, 'learning_
rate': 0.01}
0.8791812419891357,0.04131247098696789 with: {'dropout_rate': 0.1, 'learning
_rate': 0.1}
0.9142901182174683,0.03371433841124087 with: {'dropout_rate': 0.2, 'learning
_rate': 0.001}
0.9123621582984924,0.028965568516149325 with: {'dropout_rate': 0.2, 'learnin
g_rate': 0.01}
0.8601019501686096,0.06628651020498567 with: {'dropout_rate': 0.2, 'learning
_rate': 0.1}
```

From the above Hyperparameters the Learning rate :- 0.001 and Drop_out :- 0.0

Tuning of Hyperparameters:- Activation Function and Kernel Initializer

In [30]:

```
# Defining the model

def create_model(activation_function,init):
    model = Sequential()
    model.add(Dense(14,input_dim = 10,kernel_initializer = init,activation = activation_fun
    model.add(Dropout(0.0))
    model.add(Dense(8,input_dim = 14,kernel_initializer = init,activation = activation_func
    model.add(Dropout(0.0))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 40,epochs = 10)

# Define the grid search parameters
activation_function = ['softmax','relu','tanh','linear']
init = ['uniform','normal','zero']

# Make a dictionary of the grid search parameters
param_grids = dict(activation_function = activation_function,init = init)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 10)
grid_result = grid.fit(X_standardized,y)
```

```
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 2/5; 8/12] END ....activation_function=tanh, init=normal; total time=
5.1s
[CV 3/5; 8/12] START activation_function=tanh, init=norma
l.....................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 3/5; 8/12] END ....activation_function=tanh, init=normal; total time=
4.8s
[CV 4/5; 8/12] START activation_function=tanh, init=norma
l.....................
```

In [31]:

```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9167506456375122, using {'activation_function': 'relu', 'init': 'no
rmal'}
0.8911508083343506,0.03746419183631931 with: {'activation_function': 'softma
x', 'init': 'uniform'}
0.8978004217147827,0.0350226121220674 with: {'activation_function': 'softma
x', 'init': 'normal'}
0.9004593372344971,0.035401049882349434 with: {'activation_function': 'softm
ax', 'init': 'zero'}
0.9164845585823059,0.023933394033186475 with: {'activation_function': 'rel
u', 'init': 'uniform'}
0.9167506456375122,0.023907363814956124 with: {'activation_function': 'rel
u', 'init': 'normal'}
0.5625389397144318,0.06284089125503536 with: {'activation_function': 'relu',
'init': 'zero'}
0.9044490694999695,0.030078699755543916 with: {'activation_function': 'tan
h', 'init': 'uniform'}
0.9108325481414795,0.025953751423252226 with: {'activation_function': 'tan
h', 'init': 'normal'}
0.5625389397144318,0.06284089125503536 with: {'activation_function': 'tanh',
'init': 'zero'}
0.9041829347610474,0.03243667509451193 with: {'activation_function': 'linea
r', 'init': 'uniform'}
0.9082388758659363,0.027123143081747753 with: {'activation_function': 'linea
r', 'init': 'normal'}
0.5625389397144318,0.06284089125503536 with: {'activation_function': 'linea
r', 'init': 'zero'}
```

From the above Hyperparameters the Activation function : relu, initializer :normal

Tuning of Hyperparameters Numbers of neurons in activation layer

In [32]:

```
# Defining the model

def create_model(neuron1,neuron2):
    model = Sequential()
    model.add(Dense(neuron1,input_dim = 10,kernel_initializer = 'normal',activation = 'relu
    model.add(Dropout(0.0))
    model.add(Dense(neuron2,input_dim = neuron1,kernel_initializer = 'normal',activation =
    model.add(Dropout(0.0))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 40,epochs = 10)

# Define the grid search parameters

neuron1 = [4,8,16]
neuron2 = [2,4,8]

# Make a dictionary of the grid search parameters

param_grids = dict(neuron1 = neuron1,neuron2 = neuron2)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 10)
grid_result = grid.fit(X_standardized,y)
```

```
4.9s
[CV 3/5; 6/9] START neuron1=8, neuron2=
8.....................................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 3/5; 6/9] END .....................neuron1=8, neuron2=8; total time=
5.2s
[CV 4/5; 6/9] START neuron1=8, neuron2=
8.....................................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
5: UserWarning: The `lr` argument is deprecated, use `learning_rate` inste
ad.
  super(Adam, self).__init__(name, **kwargs)

[CV 4/5; 6/9] END .....................neuron1=8, neuron2=8; total time=
```

In [33]:

```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9162844300270081, using {'neuron1': 16, 'neuron2': 2}
0.9089705109596252,0.02630115112323345 with: {'neuron1': 4, 'neuron2': 2}
0.9097022175788879,0.03341632190515148 with: {'neuron1': 4, 'neuron2': 4}
0.909766960144043,0.026763335819303027 with: {'neuron1': 4, 'neuron2': 8}
0.9139570355415344,0.028742648786458942 with: {'neuron1': 8, 'neuron2': 2}
0.9110323071479798,0.0244941708636716 with: {'neuron1': 8, 'neuron2': 4}
0.9124947667121888,0.024427342372381648 with: {'neuron1': 8, 'neuron2': 8}
0.9162844300270081,0.024655817263109992 with: {'neuron1': 16, 'neuron2': 2}
0.9093700885772705,0.0361293925356696 with: {'neuron1': 16, 'neuron2': 4}
0.9120294332504273,0.033413904861325894 with: {'neuron1': 16, 'neuron2': 8}
```

Best Hyperparameters is :- neuron1 : 16 , neuron2 :2

Training model with optimum values of Hyperparameters

In [34]:

```
from sklearn.metrics import classification_report, accuracy_score

# Defining the model

def create_model():
    model = Sequential()
    model.add(Dense(16,input_dim = 10,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(0.0))
    model.add(Dense(2,input_dim = 16,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(0.0))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001) #sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rat
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 40,epochs = 10)

# Fitting the model

model.fit(X_standardized,y)

# Predicting using trained model

y_predict = model.predict(X_standardized)

# Printing the metrics
print(accuracy_score(y,y_predict))
```

```
<ipython-input-34-902dd2b3cab7>:19: DeprecationWarning: KerasClassifier is d
eprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 4
0,epochs = 10)

0.9216703238247224
```

We have got 92.1 % accuracy

Model building with train test split

In [35]:

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from keras.layers import Dropout
# Defining the model

def create_model():
    model = Sequential()
    model.add(Dense(16,input_dim = 10,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(0.0))
    model.add(Dense(2,input_dim = 16,kernel_initializer = 'normal',activation = 'relu'))
    model.add(Dropout(0.0))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001) #sgd = SGD(lr=learning_rate, momentum=momentum, decay=decay_rat
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 40,epochs = 10)

# Fitting the model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 4

model.fit(X_train,y_train)

# Predicting using trained model

y_predict = model.predict(X_test)

# Printing the metrics
print(accuracy_score( y_test ,y_predict))
```

```
<ipython-input-35-d338eef8e21c>:20: DeprecationWarning: KerasClassifier is d
eprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 4
0,epochs = 10)
C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)

0.5591755319148937
```

Accuracy : 56 %

In [36]:

```
history=model.fit(X,y,validation_split=0.2,epochs=10,batch_size=40)
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:105:
UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super(Adam, self).__init__(name, **kwargs)
```

In [37]:

```
history.history.keys()
```

Out[37]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [38]:

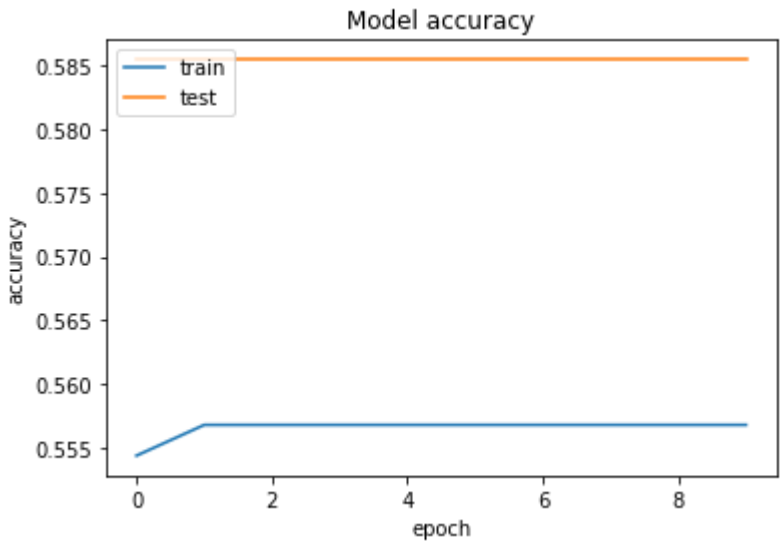
```
import matplotlib.pyplot as plt
```

In [39]:

```
# summarize history for accuracy
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.title("Model accuracy")
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['train', 'test'], loc='upper left')
```

Out[39]:

```
<matplotlib.legend.Legend at 0xd1ca796e50>
```

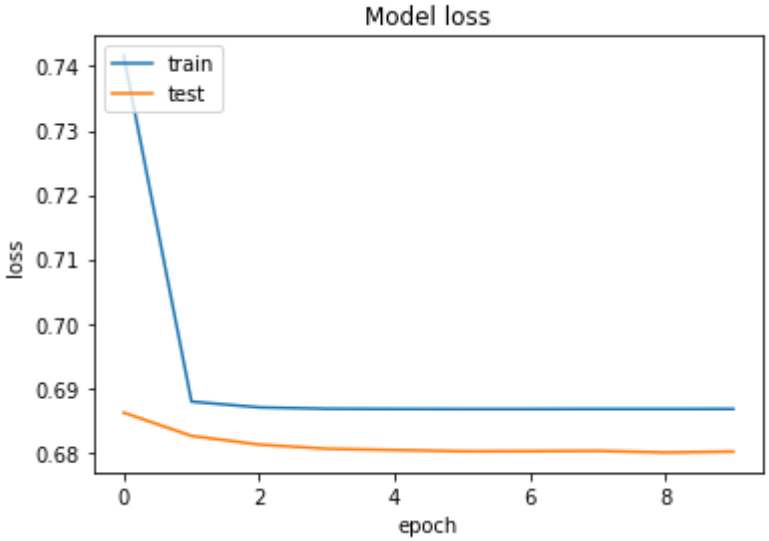


In [40]:

```
# summarize history for loss
plt.xlabel("epoch")
plt.ylabel("loss")
plt.title("Model loss")
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['train', 'test'], loc='upper left')
```

Out[40]:

```
<matplotlib.legend.Legend at 0xd1ca8fa160>
```



In [41]:

```
print("Accuracy of model: "+str(np.mean(history.history['val_accuracy'])))
print("Loss of model: "+str(np.mean(history.history['val_loss'])))
```

```
Accuracy of model: 0.5854388475418091
Loss of model: 0.6811307072639465
```

In [ ]: