Assignment-16-Neural-Network-01-(Forestfires)

Quick info. about Neural Network.

Info. :-

Neural Networks is one of the most popular machine learning algorithms.Gradient Descent forms the basis of Neural networks. Neural networks can be implemented in both R and Python using certain libraries and packages.

In [1]:

```
# Importing the libraries
import pandas as pd
import numpy as np
import keras
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder,LabelEncoder
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import GridSearchCV, KFold
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.optimizers import Adam
```

In [2]:

```
# loading dataset
forestfires=pd.read_csv("C:/Users/LENOVO/Documents/Custom Office Templates/forestfires.csv"
```

In [3]:

```
forestfires.head()
```

Out[3]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	mont
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	

5 rows × 31 columns

Exploring the dataset more

In [4]:

```
# some mathematical caculation

forestfires.describe()
```

Out[4]:

	FFMC	DMC	DC	ISI	temp	RH	wind	
count	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	51
mean	90.644681	110.872340	547.940039	9.021663	18.889168	44.288201	4.017602	
std	5.520111	64.046482	248.066192	4.559477	5.806625	16.317469	1.791653	
min	18.700000	1.100000	7.900000	0.000000	2.200000	15.000000	0.400000	
25%	90.200000	68.600000	437.700000	6.500000	15.500000	33.000000	2.700000	
50%	91.600000	108.300000	664.200000	8.400000	19.300000	42.000000	4.000000	
75%	92.900000	142.400000	713.900000	10.800000	22.800000	53.000000	4.900000	
max	96.200000	291.300000	860.600000	56.100000	33.300000	100.000000	9.400000	

8 rows × 28 columns

In [5]:

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

forestfires.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 31 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   month          517 non-null    object 
 1   day            517 non-null    object 
 2   FFMC           517 non-null    float64
 3   DMC            517 non-null    float64
 4   DC             517 non-null    float64
 5   ISI            517 non-null    float64
 6   temp           517 non-null    float64
 7   RH             517 non-null    int64  
 8   wind           517 non-null    float64
 9   rain           517 non-null    float64
 10  area           517 non-null    float64
 11  dayfri         517 non-null    int64  
 12  daymon         517 non-null    int64  
 13  daysat         517 non-null    int64  
 14  daysun         517 non-null    int64  
 15  daythu         517 non-null    int64  
 16  daytue         517 non-null    int64  
 17  daywed         517 non-null    int64  
 18  monthapr       517 non-null    int64  
 19  monthaug       517 non-null    int64  
 20  monthdec       517 non-null    int64  
 21  monthfeb       517 non-null    int64  
 22  monthjan       517 non-null    int64  
 23  monthjul       517 non-null    int64  
 24  monthjun       517 non-null    int64  
 25  monthmar       517 non-null    int64  
 26  monthmay       517 non-null    int64  
 27  monthnov       517 non-null    int64  
 28  monthoct       517 non-null    int64  
 29  monthsep       517 non-null    int64  
 30  size_category  517 non-null    object 
dtypes: float64(8), int64(20), object(3)
memory usage: 125.3+ KB
```

In [6]:

```
forestfires.shape
```

Out[6]:

```
(517, 31)
```

Data analysis

In [7]:

```
# finding NA values null values

forestfires.isna().sum()
```

Out[7]:

```
month           0
day             0
FFMC            0
DMC             0
DC              0
ISI             0
temp            0
RH              0
wind            0
rain            0
area            0
dayfri          0
daymon          0
daysat          0
daysun          0
daythu          0
daytue          0
daywed          0
monthapr        0
monthaug        0
monthdec        0
monthfeb        0
monthjan        0
monthjul        0
monthjun        0
monthmar        0
monthmay        0
monthnov        0
monthoct        0
monthsep        0
size_category   0
dtype: int64
```

No. Na \ null values present the dataset

In [8]:

```
# making the dataset safe

forest_fires = forestfires.copy()
```

In [9]:

```
forest_fires.head()
```

Out[9]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	mont
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	

5 rows × 31 columns

Data cleaning

In [10]:

```
# Creating Dummy variables for object type data

from sklearn import preprocessing
le=preprocessing.LabelEncoder()
def label_encoding(data):

    for column_name in data.columns:
        if data[column_name].dtype == object:
            data[column_name] = le.fit_transform(data[column_name])
        else:
            pass
```

In [11]:

```
label_encoding(forest_fires)
```

In [12]:

```
forest_fires.head()
```

Out[12]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	mont
0	7	0	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	
1	10	5	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	
2	10	2	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	
3	7	0	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	
4	7	3	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	

5 rows × 31 columns

Deviding the data into dependent and independent variables

In [13]:

```
X = forest_fires.iloc[: , :-1]
y = forest_fires['size_category']
```

In [14]:

```
X.head()
```

Out[14]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthdec	monthfeb	mont
0	7	0	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	
1	10	5	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	
2	10	2	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	
3	7	0	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	
4	7	3	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	

5 rows × 30 columns

Tuning of Hyperparameters :- Batch Size and Epochs

In [15]:

```
# create model
def create_model():
    model = Sequential()
    model.add(Dense(14, input_dim=30, 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 [16]:

```
# 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,y)
```

```
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 .................batch_size=20, epochs=100; total time=
4.1s
[CV 5/5; 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)

[CV 5/5; 6/9] END .................batch_size=20, epochs=100; total time=
4.4s
[CV 1/5; 7/9] START batch_size=40, epochs=1
0.................................

C:\Users\LENOVO\anaconda3\lib\site-packages\keras\optimizer_v2\adam.py:10
```

In [17]:

```
# 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 : 0.9845033645629883, using {'batch_size': 10, 'epochs': 50}
0.928192675113678,0.05199534300083278 with: {'batch_size': 10, 'epochs': 10}
0.9845033645629883,0.014520910083151746 with: {'batch_size': 10, 'epochs': 5
0}
0.8757468223571777,0.17812435864323922 with: {'batch_size': 10, 'epochs': 10
0}
0.9632001519203186,0.029666969295014924 with: {'batch_size': 20, 'epochs': 1
0}
0.9806011915206909,0.020374163551213922 with: {'batch_size': 20, 'epochs': 5
0}
0.984466016292572,0.015774825855992516 with: {'batch_size': 20, 'epochs': 10
0}
0.9670836448669433,0.019919722533084033 with: {'batch_size': 40, 'epochs': 1
0}
0.9651792287826538,0.02838736797617786 with: {'batch_size': 40, 'epochs': 5
0}
0.9767177104949951,0.026491510636519597 with: {'batch_size': 40, 'epochs': 1
00}
```

From the above best Hyperparameters :- Batch Size= 10 and Epochs = 50

Tuning of Hyperparameters:- Learning rate and Drop out rate

In [18]:

```
from keras.layers import Dropout

# Defining the model

def create_model(learning_rate,dropout_rate):
    model = Sequential()
    model.add(Dense(14,input_dim = 30,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 = 10,epochs = 50)

# 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,y)
```

```
<ipython-input-18-bf38cf0a9988>:19: DeprecationWarning: KerasClassifier is
deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size =
10,epochs = 50)
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)

Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START dropout_rate=0.0, learning_rate=0.00
1.....................
[CV 1/5; 1/9] END .....dropout_rate=0.0, learning_rate=0.001; total time=
4.3s
[CV 2/5; 1/9] START dropout_rate=0.0, learning_rate=0.00
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.
```

In [19]:

```
# 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 : 0.9786781072616577, using {'dropout_rate': 0.1, 'learning_rate': 0.00
1}
0.9593353271484375,0.03489611428187225 with: {'dropout_rate': 0.0, 'learning
_rate': 0.001}
0.9689693808555603,0.029717755122465255 with: {'dropout_rate': 0.0, 'learnin
g_rate': 0.01}
0.7325242757797241,0.15178268966445949 with: {'dropout_rate': 0.0, 'learning
_rate': 0.1}
0.9786781072616577,0.02162587770358475 with: {'dropout_rate': 0.1, 'learning
_rate': 0.001}
0.9574869275093079,0.0367941408629824 with: {'dropout_rate': 0.1, 'learning_
rate': 0.01}
0.7305825233459473,0.15435061319000673 with: {'dropout_rate': 0.1, 'learning
_rate': 0.1}
0.9650672197341919,0.04454397624999942 with: {'dropout_rate': 0.2, 'learning
_rate': 0.001}
0.9301157593727112,0.10228047758920611 with: {'dropout_rate': 0.2, 'learning
_rate': 0.01}
0.7305825233459473,0.15435061319000673 with: {'dropout_rate': 0.2, 'learning
_rate': 0.1}
```

From the above the best Hyperparameters:- Learning rate = 0.001 and Drop out rate = 0.1

Tuning of Hyperparameters:- Activation Function and Kernel Initializer

In [20]:

```
# Defining the model

def create_model(activation_function,init):
    model = Sequential()
    model.add(Dense(14,input_dim = 30,kernel_initializer = init,activation = activation_fun
    model.add(Dropout(0.1))
    model.add(Dense(8,input_dim = 14,kernel_initializer = init,activation = activation_func
    model.add(Dropout(0.1))
    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 = 10,epochs = 50)

# 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,y)
```

```
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 5/5; 3/12] END ...activation_function=softmax, init=zero; total time=
4.2s
[CV 1/5; 4/12] START activation_function=relu, init=unifor
m.....................

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 1/5; 4/12] END ...activation_function=relu, init=uniform; total time=
4.6s
[CV 2/5; 4/12] START activation_function=relu, init=unifor
m
```

In [21]:

```
# 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 : 0.9806012034416198, using {'activation_function': 'tanh', 'init': 'no
rmal'}
0.968932044506073,0.033294029695860344 with: {'activation_function': 'softma
x', 'init': 'uniform'}
0.9728342056274414,0.023315000420261068 with: {'activation_function': 'softm
ax', 'init': 'normal'}
0.8699589252471924,0.17506987527879062 with: {'activation_function': 'softma
x', 'init': 'zero'}
0.9767363667488098,0.01801522501153033 with: {'activation_function': 'relu',
'init': 'uniform'}
0.9747759461402893,0.02719517406905011 with: {'activation_function': 'relu',
'init': 'normal'}
0.7305825233459473,0.15435061319000673 with: {'activation_function': 'relu',
'init': 'zero'}
0.9554144740104675,0.04147513583699858 with: {'activation_function': 'tanh',
'init': 'uniform'}
0.9806012034416198,0.016257026245678638 with: {'activation_function': 'tan
h', 'init': 'normal'}
0.7305825233459473,0.15435061319000673 with: {'activation_function': 'tanh',
'init': 'zero'}
0.9786407709121704,0.026339153500535555 with: {'activation_function': 'linea
r', 'init': 'uniform'}
0.974794614315033,0.02092379439916338 with: {'activation_function': 'linea
r', 'init': 'normal'}
0.7305825233459473,0.15435061319000673 with: {'activation_function': 'linea
r', 'init': 'zero'}
```

From the above the best Hyperparameters:- Activation Function = 'tanh' and Kernel Initializer = 'normal'

Tuning of Hyperparameter :-Number of Neurons in activation layer

In [22]:

```
# Defining the model

def create_model(neuron1,neuron2):
    model = Sequential()
    model.add(Dense(neuron1,input_dim = 30,kernel_initializer = 'normal',activation = 'tanh
    model.add(Dropout(0.1))
    model.add(Dense(neuron2,input_dim = neuron1,kernel_initializer = 'normal',activation =
    model.add(Dropout(0.1))
    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 = 10,epochs = 50)

# 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,y)
```

```
<ipython-input-22-12b1844d38d4>:17: DeprecationWarning: KerasClassifier is
deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size =
10,epochs = 50)
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)

Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START neuron1=4, neuron2=
2.....................................
[CV 1/5; 1/9] END .....................neuron1=4, neuron2=2; total time=
3.9s
[CV 2/5; 1/9] START neuron1=4, neuron2=
2.....................................

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.
```

In [23]:

```
# 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 : 0.9787341237068177, using {'neuron1': 4, 'neuron2': 2}
0.9787341237068177,0.018691590022879234 with: {'neuron1': 4, 'neuron2': 2}
0.9728342056274414,0.0365427156757491 with: {'neuron1': 4, 'neuron2': 4}
0.9766990184783936,0.028537757616977678 with: {'neuron1': 4, 'neuron2': 8}
0.9670089721679688,0.04411790327817285 with: {'neuron1': 8, 'neuron2': 2}
0.9729275584220887,0.023952627734300513 with: {'neuron1': 8, 'neuron2': 4}
0.9786967992782593,0.0177722916418999 with: {'neuron1': 8, 'neuron2': 8}
0.9670276284217835,0.020017025600638026 with: {'neuron1': 16, 'neuron2': 2}
0.9748693108558655,0.022481687688530344 with: {'neuron1': 16, 'neuron2': 4}
0.9766990184783936,0.025017672694543486 with: {'neuron1': 16, 'neuron2': 8}
```

From the above best Hyperparameter :-Number of Neurons in activation layer - Neuron1 = 4 and Neuron 2 = 2

Training model with optimum values of Hyperparameters

In [24]:

```
from sklearn.metrics import classification_report, accuracy_score

# Defining the model

def create_model():
    model = Sequential()
    model.add(Dense(4,input_dim = 30,kernel_initializer = 'normal',activation = 'tanh'))
    model.add(Dropout(0.1))
    model.add(Dense(2,input_dim = 4,kernel_initializer = 'normal',activation = 'tanh'))
    model.add(Dropout(0.1))
    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 = 10,epochs = 50)

# Fitting the model

model.fit(X,y)

# Predicting using trained model

y_predict = model.predict(X)

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

```
<ipython-input-24-166423463d96>: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 = 1
0,epochs = 50)
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)

0.988394584139265
```

After Tunning the model with hyperparameters we have got accuracy of 99.0 %

let us split the data into train and test and see the results

In [25]:

```
from sklearn.model_selection import train_test_split


from sklearn.metrics import classification_report, accuracy_score

# Defining the model

def create_model():
    model = Sequential()
    model.add(Dense(4,input_dim = 30,kernel_initializer = 'normal',activation = 'tanh'))
    model.add(Dropout(0.1))
    model.add(Dense(2,input_dim = 4,kernel_initializer = 'normal',activation = 'tanh'))
    model.add(Dropout(0.1))
    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 = 10,epochs = 50)

# Fitting the model

X_train, X_test,y_train,y_test = train_test_split(X,y, test_size=0.2, random_state=40 )
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-25-f2a382ff7cf8>:22: DeprecationWarning: KerasClassifier is d
eprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead.
  model = KerasClassifier(build_fn = create_model,verbose = 0,batch_size = 1
0,epochs = 50)
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)

0.9519230769230769
```

We have accuracy of 95 % from the above tuned hyperparameter model

In [26]:

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

In [28]:

```
history=model.fit(X,y,validation_split=0.2,epochs=50,batch_size=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)
```

In [29]:

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

Out[29]:

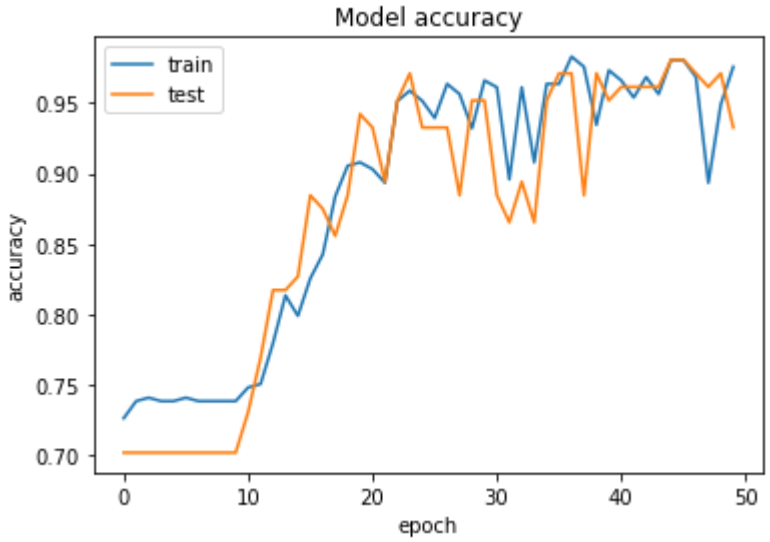
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [30]:

```
# 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[30]:

```
<matplotlib.legend.Legend at 0x6ba4d39550>
```

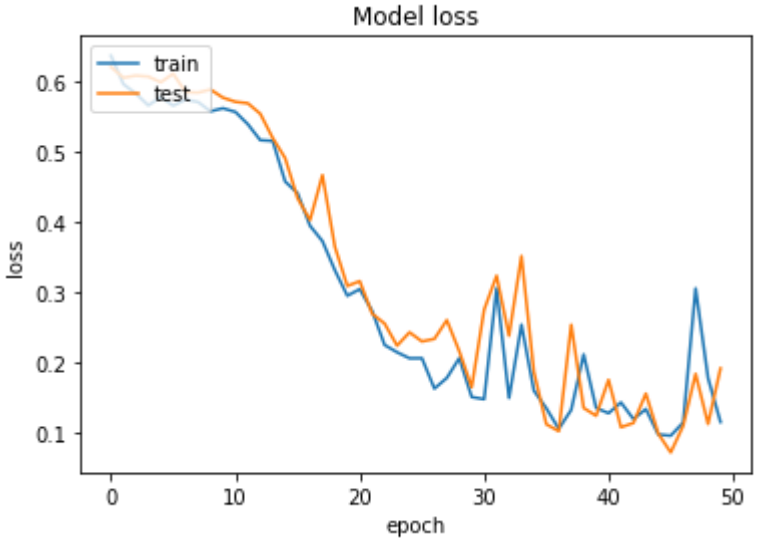


In [31]:

```
# 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[31]:

```
<matplotlib.legend.Legend at 0x6ba0db21c0>
```



In [32]:

```
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: 0.8723076868057251
Loss of model: 0.32981587648391725
```

In [ ]: