In [1]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /opt/anaconda3/lib/py
thon3.9/site-packages (2.8.0)
Requirement already satisfied: numpy>=1.20 in /opt/anaconda3/lib/p
ython3.9/site-packages (from tensorflow) (1.20.3)
Requirement already satisfied: flatbuffers>=1.12 in /opt/anaconda3
/lib/python3.9/site-packages (from tensorflow) (2.0)
Requirement already satisfied: h5py>=2.9.0 in /opt/anaconda3/lib/p
ython3.9/site-packages (from tensorflow) (3.2.1)
Requirement already satisfied: wrapt>=1.11.0 in /opt/anaconda3/lib
/python3.9/site-packages (from tensorflow) (1.12.1)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /opt/anacon
da3/lib/python3.9/site-packages (from tensorflow) (2.8.0)
Requirement already satisfied: protobuf>=3.9.2 in /opt/anaconda3/l
ib/python3.9/site-packages (from tensorflow) (3.19.4)
Requirement already satisfied: setuptools in /opt/anaconda3/lib/py
thon3.9/site-packages (from tensorflow) (58.0.4)
Requirement already satisfied: libclang>=9.0.1 in /opt/anaconda3/l
ib/python3.9/site-packages (from tensorflow) (13.0.0)
Requirement already satisfied: tensorboard<2.9,>=2.8 in /opt/anaco
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_sco
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
from sklearn.decomposition import PCA
import tensorflow as tf
tf.debugging.set_log_device_placement(False)
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
tf.random.set_seed(14)
```

In [4]:
```
raw_data = pd.read_csv("forestfires.csv")
raw_data.head()
```

Out[4]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	r
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

In [5]:
```
df = raw_data.copy() #Removing the dummies at this time
df.drop(df.columns[11:30],axis=1,inplace = True)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 12 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  size_category  517 non-null    object 
dtypes: float64(8), int64(1), object(3)
memory usage: 48.6+ KB
```

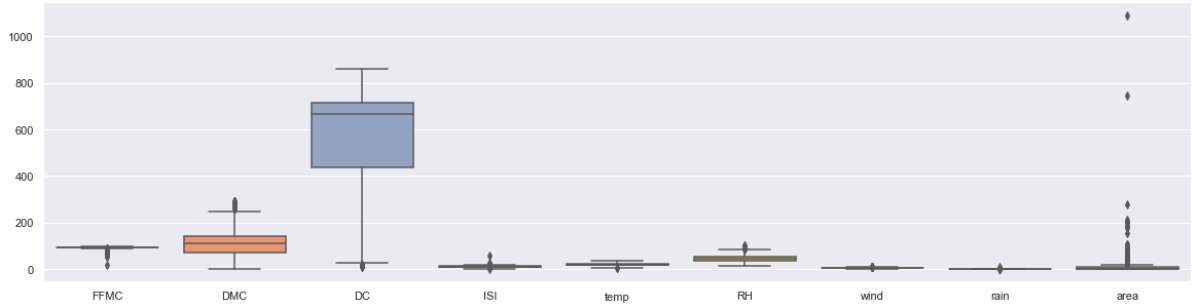
In [7]:
```
df.describe()
```

Out[7]:

	FFMC	DMC	DC	ISI	temp	RH	wind
count	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000	517.000000
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

In [8]:
```
sns.set(rc={'figure.figsize':(20,5)})
sns.boxplot(data=df, orient="v", palette="Set2")
```

Out[8]: <AxesSubplot:>



Feature Analysis

In [9]:
```
df.month.value_counts()
```

Out[9]:
```
aug    184
sep    172
mar     54
jul     32
feb     20
jun     17
oct     15
apr      9
dec      9
jan      2
may      2
nov      1
Name: month, dtype: int64
```

In [10]:
```
df.size_category.value_counts()
```

Out[10]:
```
small    378
large    139
Name: size_category, dtype: int64
```

In [11]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
df.month= label_encoder.fit_transform(df.month)
df.day= label_encoder.fit_transform(df.day)
df.size_category= label_encoder.fit_transform(df.size_category)
df.head()
```

Out[11]:

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

Removing Bias in the Dataset

In [12]:
```
pip install imblearn
```

```
Requirement already satisfied: imblearn in /opt/anaconda3/lib/pyth
on3.9/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/anaconda3/
lib/python3.9/site-packages (from imblearn) (0.10.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anacon
da3/lib/python3.9/site-packages (from imbalanced-learn->imblearn)
(2.2.0)
Requirement already satisfied: joblib>=1.1.1 in /opt/anaconda3/lib
/python3.9/site-packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /opt/anaconda3/lib/
python3.9/site-packages (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /opt/anacond
a3/lib/python3.9/site-packages (from imbalanced-learn->imblearn) (
1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib
/python3.9/site-packages (from imbalanced-learn->imblearn) (1.20.3
)
Note: you may need to restart the kernel to use updated packages.
```

In [13]:
```
from imblearn.combine import SMOTETomek
from collections import Counter

resamp = df.copy()

a = resamp.iloc[:,:-1]
b = resamp.iloc[:,-1]

print(Counter(b))

smt = SMOTETomek(sampling_strategy = 'auto')
a, b = smt.fit_resample(a, b)

print(Counter(b))
```

```
Counter({1: 378, 0: 139})
Counter({1: 371, 0: 371})
```

Train | Split dataset

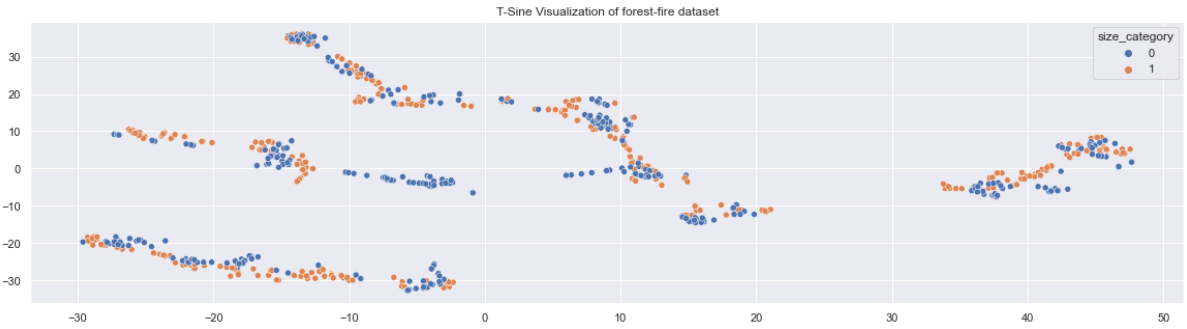
In [14]:
```
n, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3)
```

Visualizing the data

In [15]:
```
#Visualising the data via dimensionality reduction t-Sine Technique
from sklearn.manifold import TSNE

data_tsne_pca = TSNE(n_components=2).fit_transform(a)
sns.scatterplot(data_tsne_pca[:,0],data_tsne_pca[:,1],hue=b, palett
```

Out[15]:
```
Text(0.5, 1.0, 'T-Sine Visualization of forest-fire dataset')
```



Neural Network Modelling

In [16]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
import keras
keras. __version__
'2.4.3'
```

Out[16]: '2.4.3'

In [17]:
```
model1 = Sequential()
model1.add(Dense(14, input_dim=11, kernel_initializer='uniform', ac
model1.add(Dense(12,kernel_initializer='uniform', activation='relu'
model1.add(Dense(1, kernel_initializer='uniform', activation='sigmo

model1.compile(loss='binary_crossentropy', optimizer='adam', metric

hist1 = model1.fit(X_train, y_train, validation_split=0.33, epochs=
```

```
3 - accuracy: 0.9885 - val_loss: 0.0439 - val_accuracy: 0.9826
Epoch 25/30
35/35 [==============================] - 0s 4ms/step - loss: 0.049
5 - accuracy: 0.9885 - val_loss: 0.0576 - val_accuracy: 0.9767
Epoch 26/30
35/35 [==============================] - 0s 4ms/step - loss: 0.042
8 - accuracy: 0.9914 - val_loss: 0.0410 - val_accuracy: 0.9942
Epoch 27/30
35/35 [==============================] - 0s 3ms/step - loss: 0.041
4 - accuracy: 0.9885 - val_loss: 0.0400 - val_accuracy: 0.9826
Epoch 28/30
35/35 [==============================] - 0s 3ms/step - loss: 0.046
3 - accuracy: 0.9885 - val_loss: 0.0416 - val_accuracy: 0.9826
Epoch 29/30
35/35 [==============================] - 0s 3ms/step - loss: 0.040
3 - accuracy: 0.9885 - val_loss: 0.0374 - val_accuracy: 0.9942
Epoch 30/30
35/35 [==============================] - 0s 3ms/step - loss: 0.079
7 - accuracy: 0.9741 - val_loss: 0.0360 - val_accuracy: 0.9884
```

Model Evaluation

In [18]:
```
test_loss, test_acc = model1.evaluate(X_test, y_test)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 0.0424
- accuracy: 0.9865
```

In [19]:
```
print(hist1.history.keys())
```

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

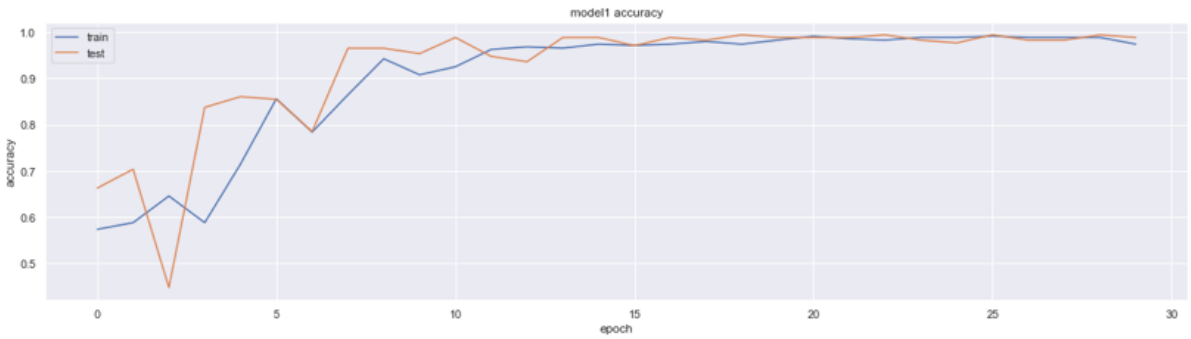
In [20]:
```
hist1_df = pd.DataFrame(hist1.history)
hist1_df["epoch"]=hist1.epoch
hist1_df.tail()
```

Out[20]:

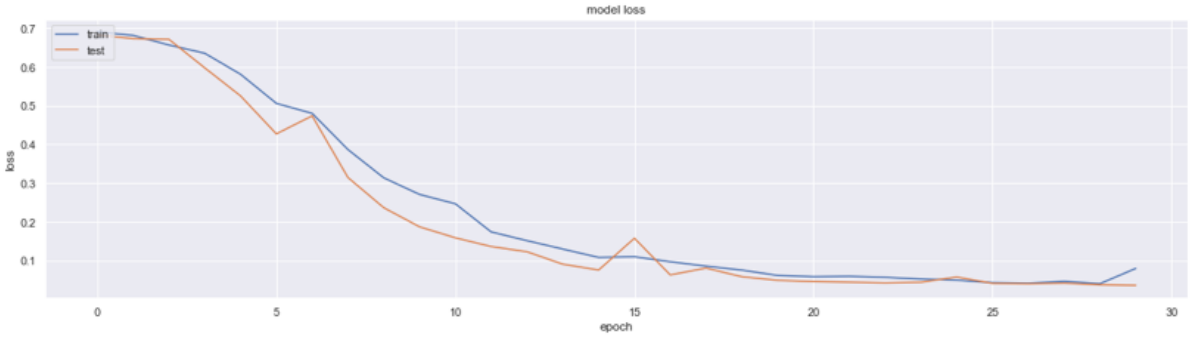
	loss	accuracy	val_loss	val_accuracy	epoch
25	0.042817	0.991354	0.040970	0.994186	25
26	0.041406	0.988473	0.040005	0.982558	26
27	0.046269	0.988473	0.041612	0.982558	27
28	0.040295	0.988473	0.037357	0.994186	28
29	0.079652	0.974063	0.035973	0.988372	29

Visualize Training History

In [21]:
```
# summarize history for accuracy
plt.plot(hist1.history['accuracy'])
plt.plot(hist1.history['val_accuracy'])
plt.title('model1 accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [22]:
```
# summarize history for loss
plt.plot(hist1.history['loss'])
plt.plot(hist1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show() #Loss decreases and starts oscillating after 25
```



Tuning HyperParameters

In [23]:
```
from sklearn.preprocessing import StandardScaler
a = StandardScaler()
a.fit(X)
X_standardized = a.transform(X)



from tensorflow import keras
from keras.layers import Conv2D, Input, MaxPool2D,Flatten, Dense, P
from keras.models import Model
from tensorflow.keras.optimizers import Adam
# Importing the necessary packages
from sklearn.model_selection import GridSearchCV, KFold
from keras.wrappers.scikit_learn import KerasClassifier
```

In [24]:
```
# create model function
def create_model():
    model = Sequential()
    model.add(Dense(14, input_dim=11, kernel_initializer='uniform',
    model.add(Dense(12, kernel_initializer='uniform', activation='r
    model.add(Dense(1, kernel_initializer='uniform', activation='si

    adam=Adam(lr=0.01) #learning rate = 0.01
    model.compile(loss='binary_crossentropy', optimizer=adam, metri
    return model
```

In [25]:
```
# Create the model
model = KerasClassifier(build_fn = create_model,verbose = 0)
# Define the grid search parameters
batch_size = [10,20,40]
epochs = [10,25,50,100]
# 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 = K
grid_result = grid.fit(X_standardized,Y)
```

```
[CV 1/5; 7/12] START batch_size=20, epochs=50.....................
..............
[CV 1/5; 7/12] END ....batch_size=20, epochs=50;, score=0.993 tota
l time=   3.7s
[CV 2/5; 7/12] START batch_size=20, epochs=50.....................
..............
[CV 2/5; 7/12] END ....batch_size=20, epochs=50;, score=0.973 tota
l time=   3.8s
[CV 3/5; 7/12] START batch_size=20, epochs=50.....................
..............
[CV 3/5; 7/12] END ....batch_size=20, epochs=50;, score=0.926 tota
l time=   3.7s
[CV 4/5; 7/12] START batch_size=20, epochs=50.....................
..............
[CV 4/5; 7/12] END ....batch_size=20, epochs=50;, score=0.959 tota
l time=   5.3s
[CV 5/5; 7/12] START batch_size=20, epochs=50.....................
..............
[CV 5/5; 7/12] END ....batch_size=20, epochs=50;, score=1.000 tota
```

In [26]:
```
# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_res
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))
#According to this, the best batch size is 20 and epochs is 50
```

```
Best : 0.9864955544471741, using {'batch_size': 10, 'epochs': 100}
0.9513967037200928,0.028515675938569962 with: {'batch_size': 10, '
epochs': 10}
0.8777253746986389,0.19508837677879967 with: {'batch_size': 10, 'e
pochs': 25}
0.9797569394111634,0.0171077493914455 with: {'batch_size': 10, 'ep
ochs': 50}
0.9864955544471741,0.014177425538819203 with: {'batch_size': 10, '
epochs': 100}
0.9446852803230286,0.03710718898889838 with: {'batch_size': 20, 'e
pochs': 10}
0.9717123150825501,0.01971930494401351 with: {'batch_size': 20, 'e
pochs': 25}
0.9703156232833863,0.026561463295143847 with: {'batch_size': 20, '
epochs': 50}
0.9784055709838867,0.01623362246352538 with: {'batch_size': 20, 'e
pochs': 100}
0.9420007228851318,0.02890655663899093 with: {'batch_size': 40, 'e
pochs': 10}
0.9743606090545655,0.02614612839383079 with: {'batch_size': 40, 'e
pochs': 25}
0.9649192810058593,0.0315288799558257 with: {'batch_size': 40, 'ep
ochs': 50}
0.9784146547317505,0.02111697180563805 with: {'batch_size': 40, 'e
pochs': 100}
```

In [27]:
```
from keras.layers import Dropout



def create_model2(learning_rate,dropout_rate):
    model = Sequential()
    model.add(Dense(14,input_dim = 11,kernel_initializer = 'uniform'
    model.add(Dropout(dropout_rate))
    model.add(Dense(12,input_dim = 14,kernel_initializer = 'uniform'
    model.add(Dropout(dropout_rate))
    model.add(Dense(1,kernel_initializer='uniform',activation = 'sig

    adam = Adam(lr = learning_rate)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metr
    return model
```

```
model2 = KerasClassifier(build_fn = create_model2,verbose = 0,batch_



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_grids2 = dict(learning_rate = learning_rate,dropout_rate = dro



grid2 = GridSearchCV(estimator = model2,param_grid = param_grids2,cv
grid_result2 = grid2.fit(X_standardized,Y)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START dropout_rate=0.0, learning_rate=0.001.........
..............
[CV 1/5; 1/9] END dropout_rate=0.0, learning_rate=0.001;, score=0.
993 total time=   3.1s
[CV 2/5; 1/9] START dropout_rate=0.0, learning_rate=0.001.........
..............
[CV 2/5; 1/9] END dropout_rate=0.0, learning_rate=0.001;, score=0.
980 total time=   3.9s
[CV 3/5; 1/9] START dropout_rate=0.0, learning_rate=0.001.........
..............
[CV 3/5; 1/9] END dropout_rate=0.0, learning_rate=0.001;, score=0.
959 total time=   5.2s
[CV 4/5; 1/9] START dropout_rate=0.0, learning_rate=0.001.........
..............
[CV 4/5; 1/9] END dropout_rate=0.0, learning_rate=0.001;, score=0.
959 total time=   2.9s
[CV 5/5; 1/9] START dropout_rate=0.0, learning_rate=0.001.........
..............
[CV 5/5; 1/9] END dropout_rate=0.0, learning_rate=0.001;, score=1
```

In [28]:
```
 {}, using {}'.format(grid_result2.best_score_,grid_result2.best_par
_result2.cv_results_['mean_test_score']
result2.cv_results_['std_test_score']
d_result2.cv_results_['params']
ev, param in zip(means2, stds2, params2):
} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9919009566307068, using {'dropout_rate': 0.2, 'learning_r
ate': 0.001}
0.9784146666526794,0.01678012811351063 with: {'dropout_rate': 0.0,
'learning_rate': 0.001}
0.9690005421638489,0.017412645987242195 with: {'dropout_rate': 0.0
, 'learning_rate': 0.01}
0.9676219820976257,0.015059663839577712 with: {'dropout_rate': 0.0
, 'learning_rate': 0.1}
0.9851441979408264,0.011631105147373604 with: {'dropout_rate': 0.1
, 'learning_rate': 0.001}
0.9783965110778808,0.016778636188025203 with: {'dropout_rate': 0.1
, 'learning_rate': 0.01}
0.9581897377967834,0.02813256180912803 with: {'dropout_rate': 0.1,
'learning_rate': 0.1}
0.9919009566307068,0.005058744039366999 with: {'dropout_rate': 0.2
, 'learning_rate': 0.001}
0.9797932028770446,0.015344378601774884 with: {'dropout_rate': 0.2
, 'learning_rate': 0.01}
0.9230818152427673,0.04750037802334864 with: {'dropout_rate': 0.2,
'learning_rate': 0.1}
```

In [29]:
```
def create_model3(activation_function,init):
    model = Sequential()
    model.add(Dense(14,input_dim = 11,kernel_initializer = init,act
    model.add(Dropout(0.2))
    model.add(Dense(12,input_dim = 14,kernel_initializer = init,act
    model.add(Dropout(0.2))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,met
    return model



model3 = KerasClassifier(build_fn = create_model3,verbose = 0,batch


activation_function = ['softmax','relu','tanh','linear']
init = ['uniform','normal','zero']

# Make a dictionary of the grid search parameters
param_grids3 = dict(activation_function = activation_function,init


grid3 = GridSearchCV(estimator = model3,param_grid = param_grids3,c
grid_result3 = grid3.fit(X_standardized,Y)
```

```
[CV 2/5; 9/12] START activation_function=tanh, init=zero.........
.............
[CV 2/5; 9/12] END activation_function=tanh, init=zero;, score=0.4
90 total time=   3.9s
[CV 3/5; 9/12] START activation_function=tanh, init=zero.........
.............
[CV 3/5; 9/12] END activation_function=tanh, init=zero;, score=0.2
84 total time=   3.0s
[CV 4/5; 9/12] START activation_function=tanh, init=zero.........
.............
[CV 4/5; 9/12] END activation_function=tanh, init=zero;, score=0.2
70 total time=   3.9s
[CV 5/5; 9/12] START activation_function=tanh, init=zero.........
.............
[CV 5/5; 9/12] END activation_function=tanh, init=zero;, score=0.0
00 total time=   3.0s
[CV 1/5; 10/12] START activation_function=linear, init=uniform....
.............
[CV 1/5; 10/12] END activation_function=linear, init=uniform;, sco
re=1.000 total time=   2.9s
```

In [31]:
```
print('Best : {}, using {}'.format(grid_result3.best_score_,grid_re
means3 = grid_result3.cv_results_['mean_test_score']
stds3 = grid_result3.cv_results_['std_test_score']
params3 = grid_result3.cv_results_['params']
for mean, stdev, param in zip(means3, stds3, params3):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9919009566307068, using {'activation_function': 'linear',
'init': 'normal'}
0.3774079442024231,0.2655184326674861 with: {'activation_function'
: 'softmax', 'init': 'uniform'}
0.38684019446372986,0.2745600657506225 with: {'activation_function
': 'softmax', 'init': 'normal'}
0.34360602367669346,0.2710875974876528 with: {'activation_function
': 'softmax', 'init': 'zero'}
0.9878469109535217,0.012389301638458257 with: {'activation_functio
n': 'relu', 'init': 'uniform'}
0.9891982555389405,0.010981791123206273 with: {'activation_functio
n': 'relu', 'init': 'normal'}
0.20879738926887512,0.18742024777163002 with: {'activation_functio
n': 'relu', 'init': 'zero'}
0.9905405402183532,0.010112579911255349 with: {'activation_functio
n': 'tanh', 'init': 'uniform'}
0.9878468990325928,0.011628994180010722 with: {'activation_functio
n': 'tanh', 'init': 'normal'}
0.20879738926887512,0.18742024777163002 with: {'activation_functio
n': 'tanh', 'init': 'zero'}
0.991891884803772,0.007879668203816833 with: {'activation_function
': 'linear', 'init': 'uniform'}
0.9919009566307068,0.005058744039366999 with: {'activation_functio
n': 'linear', 'init': 'normal'}
0.20879738926887512,0.18742024777163002 with: {'activation_functio
n': 'linear', 'init': 'zero'}
```

In [32]:
```
def create_model4(neuron1,neuron2):
    model = Sequential()
    model.add(Dense(neuron1,input_dim = 11,kernel_initializer = 'un
    model.add(Dropout(0.2))
    model.add(Dense(neuron2,input_dim = neuron1,kernel_initializer
    model.add(Dropout(0.2))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,met
    return model


model4 = KerasClassifier(build_fn = create_model4,verbose = 0,batch


neuron1 = [4,8,14]
neuron2 = [4,8,12]

# Make a dictionary of the grid search parameters

param_grids4 = dict(neuron1 = neuron1,neuron2 = neuron2)

grid4 = GridSearchCV(estimator = model4,param_grid = param_grids4,c
grid_result4 = grid4.fit(X_standardized,Y)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START neuron1=4, neuron2=4.........................
..............
[CV 1/5; 1/9] END .........neuron1=4, neuron2=4;, score=1.000 tota
l time=   3.0s
[CV 2/5; 1/9] START neuron1=4, neuron2=4.........................
..............
[CV 2/5; 1/9] END .........neuron1=4, neuron2=4;, score=0.993 tota
l time=   2.5s
[CV 3/5; 1/9] START neuron1=4, neuron2=4.........................
..............
[CV 3/5; 1/9] END .........neuron1=4, neuron2=4;, score=0.966 tota
l time=   2.6s
[CV 4/5; 1/9] START neuron1=4, neuron2=4.........................
..............
[CV 4/5; 1/9] END .........neuron1=4, neuron2=4;, score=0.993 tota
l time=   2.5s
[CV 5/5; 1/9] START neuron1=4, neuron2=4.........................
..............
[CV 5/5; 1/9] END
```

In [33]:
```
'Best : {}, using {}'.format(grid_result4.best_score_,grid_result4.b
 = grid_result4.cv_results_['mean_test_score']
= grid_result4.cv_results_['std_test_score']
4 = grid_result4.cv_results_['params']
an, stdev, param in zip(means4, stds4, params4):
t('{},{} with: {}'.format(mean, stdev, param))
```

```
Best : 0.9838019251823426, using {'neuron1': 14, 'neuron2': 4}
0.9432523012161255,0.09061611062948016 with: {'neuron1': 4, 'neuro
n2': 4}
0.9689189314842224,0.03243242353200964 with: {'neuron1': 4, 'neuro
n2': 8}
0.9810992121696472,0.01564932069519009 with: {'neuron1': 4, 'neuro
n2': 12}
0.9662343502044678,0.03524939211464407 with: {'neuron1': 8, 'neuro
n2': 4}
0.98244149684906,0.012539749867543805 with: {'neuron1': 8, 'neuron
2': 8}
0.9811173558235169,0.00990096235815999 with: {'neuron1': 8, 'neuro
n2': 12}
0.9838019251823426,0.015180191410213317 with: {'neuron1': 14, 'neu
ron2': 4}
0.9810901403427124,0.013789149485065716 with: {'neuron1': 14, 'neu
ron2': 8}
0.9824596285820008,0.012545636900063844 with: {'neuron1': 14, 'neu
ron2': 12}
```

Building Final Model

In [34]:
```
def create_model_fnl():
    model = Sequential()
    model.add(Dense(4,input_dim = 11,kernel_initializer = 'uniform'
    model.add(Dropout(0.2))
    model.add(Dense(4,input_dim = 4,kernel_initializer = 'uniform',
    model.add(Dropout(0.2))
    model.add(Dense(1,activation = 'sigmoid'))

    adam = Adam(lr = 0.001) #sgd = SGD(lr=learning_rate, momentum=m
    model.compile(loss = 'binary_crossentropy',optimizer = adam,met
    return model
```

In [35]:
```
model_fnl = KerasClassifier(build_fn = create_model_fnl,verbose = 0
```

In [36]:
```
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_stand
```

In [37]:
```
# Fitting the model
hist2 = model_fnl.fit(X_train_1,y_train_1)
```

In [39]:
```
pip install --upgrade scikit-learn
```

```
Requirement already satisfied: scikit-learn in /opt/anaconda3/lib/
python3.9/site-packages (1.2.0)
Requirement already satisfied: scipy>=1.3.2 in /opt/anaconda3/lib/
python3.9/site-packages (from scikit-learn) (1.7.1)
Requirement already satisfied: joblib>=1.1.1 in /opt/anaconda3/lib
/python3.9/site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anacon
da3/lib/python3.9/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: numpy>=1.17.3 in /opt/anaconda3/lib
/python3.9/site-packages (from scikit-learn) (1.20.3)
Note: you may need to restart the kernel to use updated packages.
```

In [42]:
```
from sklearn.metrics import confusion_matrix,classification_report
def report_model(model):
    model_preds = model.predict(X_test_1)
    print(confusion_matrix(y_test_1,model_preds))
    print(classification_report(y_test_1,model_preds))
report_model(model_fnl)
```

```
[[112   3]
 [  2 106]]
              precision    recall  f1-score   support

           0       0.98      0.97      0.98       115
           1       0.97      0.98      0.98       108

    accuracy                           0.98       223
   macro avg       0.98      0.98      0.98       223
weighted avg       0.98      0.98      0.98       223
```

In [ ]:
```

```