In [1]:
```python
# Import required libraries
import tensorflow as tf
import matplotlib as mpl
import numpy as np
import pandas as pd
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import Sequential, regularizers
from tensorflow.keras.models import Model
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.manifold import Isomap
```

In [2]:
```python
X = pd.read_csv('parkinsons.data')
X.head()
```

Out[2]:

| | name | MDVP:Fo(Hz) | MDVP:Fhi(Hz) | MDVP:Flo(Hz) | MDVP:Jitter(%) | MDVP:Jitter(Abs) | MDVP:RAP | MDVP:PPQ | Jitter:DDP | MDVP:Shimmer | MDVP:Shimmer(dB) | Shimmer:A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | phon_R01_S01_1 | 119.992 | 157.302 | 74.997 | 0.00784 | 0.00007 | 0.00370 | 0.00554 | 0.01109 | 0.04374 | 0.426 | 0.0 |
| 1 | phon_R01_S01_2 | 122.400 | 148.650 | 113.819 | 0.00968 | 0.00008 | 0.00465 | 0.00696 | 0.01394 | 0.06134 | 0.626 | 0.0 |
| 2 | phon_R01_S01_3 | 116.682 | 131.111 | 111.555 | 0.01050 | 0.00009 | 0.00544 | 0.00781 | 0.01633 | 0.05233 | 0.482 | 0.0 |
| 3 | phon_R01_S01_4 | 116.676 | 137.871 | 111.366 | 0.00997 | 0.00009 | 0.00502 | 0.00698 | 0.01505 | 0.05492 | 0.517 | 0.0 |
| 4 | phon_R01_S01_5 | 116.014 | 141.781 | 110.655 | 0.01284 | 0.00011 | 0.00655 | 0.00908 | 0.01966 | 0.06425 | 0.584 | 0.0 |

In [3]:
```python
# Drop the `name` column
X.drop(axis = 1, labels = ['name'], inplace = True)
```

In [4]:
```python
# Extract the label for the data
Y = X.pop('status')
```

In [5]:
```python
# Using sklearn split the data into train and test slits. 20% test and 80% train
from sklearn.model_selection import train_test_split as tts
x_train, x_test, y_train, y_test = tts(X,Y, test_size = .2, random_state = 12345)
```

In [6]:
```python
# Scale the training data
from sklearn import preprocessing
scale = preprocessing.StandardScaler().fit(x_train)
```

In [7]:
```python
# Apply scale to training and test data
x_train, x_test = scale.transform(x_train), scale.transform(x_test)
```

In [8]:
```python
print(f'Label: {y_train.unique()}\n1: Has Parkinson \n0: Does not have Parkinsons')
```

```
Label: [1 0]
1: Has Parkinson
0: Does not have Parkinsons
```

In [9]:
```python
# Convert label to one-hot
from tensorflow.keras.utils import to_categorical
y_train,y_test = to_categorical(y_train), to_categorical(y_test)
```

In [9]:

## Predict with neural network

In [10]:
```python
def classifia(layers):
    """layers: list of the dimensions of each layer of network"""
    X = Input(x_train.shape[1],)
    model = Sequential()(X)
    model = Dense(layers[0], activation = 'relu')(model)
    if len(layers) > 2:
        for layer in layers[1:-1]:
            model = Dense(layer, activation = 'relu')(model)
    model = Dense(layers[-1], activation = 'softmax')(model)
    model = Model(inputs = X, outputs = model, name=f'classifier{len(layers)}')
    model.compile(loss = 'categorical_crossentropy', optimizer = 'Adam', metrics = ['accuracy'])
    return model
```

In [11]:
```python
layers = [2048,2048,1024, y_train.shape[-1]]
model = classifia(layers)
```

In [12]:
```python
model.summary()
```

```
Model: "classifier4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 22)]              0
_________________________________________________________________
sequential (Sequential)      multiple                  0
_________________________________________________________________
dense (Dense)                (None, 2048)              47104
_________________________________________________________________
dense_1 (Dense)              (None, 2048)              4196352
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              2098176
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 2050
=================================================================
Total params: 6,343,682
Trainable params: 6,343,682
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```python
m = model.fit(x_train, y_train, epochs = 100, verbose = 1)
```

```
Epoch 1/100
5/5 [==============================] - 0s 7ms/step - loss: 0.5018 - accuracy: 0.6667
Epoch 2/100
5/5 [==============================] - 0s 7ms/step - loss: 0.3109 - accuracy: 0.8462
Epoch 3/100
5/5 [==============================] - 0s 7ms/step - loss: 0.2140 - accuracy: 0.9038
Epoch 4/100
5/5 [==============================] - 0s 8ms/step - loss: 0.1613 - accuracy: 0.9423
Epoch 5/100
5/5 [==============================] - 0s 7ms/step - loss: 0.1636 - accuracy: 0.9231
Epoch 6/100
5/5 [==============================] - 0s 6ms/step - loss: 0.1073 - accuracy: 0.9295
Epoch 7/100
5/5 [==============================] - 0s 6ms/step - loss: 0.0774 - accuracy: 0.9808
Epoch 8/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0781 - accuracy: 0.9679
Epoch 9/100
5/5 [==============================] - 0s 7ms/step - loss: 0.0379 - accuracy: 0.9872
Epoch 10/100
```

In [ ]:
```python
pred = model.predict(x_test)
```

In [15]:
```python
print(pd.Series([np.argmax(y) for y in y_test]).value_counts())
print('Null Accuracy: ',str(round(pd.Series([np.argmax(y) for y in y_test]).value_counts().head(1)[1]/len(y_test) * 100))+'%')
print(f'Test Accuracy: {metrics.accuracy_score(y_test, np.where(pred>=.5, 1, 0)) * 100}')
print(f'Train Accuracy: {metrics.accuracy_score(y_train, np.where(model.predict(x_train)>=.5, 1, 0)) * 100}')

print()
# Confusion matrix
test = np.array([np.argmax(y) for y in y_test])
prediction = np.array([np.argmax(y) for y in np.where(model.predict(x_test)>=.5, 1, 0)])
confusion = metrics.confusion_matrix(test, prediction); print(f'Confusion:\n{confusion}')
```

```
1    29
0    10
dtype: int64
Null Accuracy:  74.0%
Test Accuracy: 94.87179487179486
Train Accuracy: 100.0

Confusion:
[[ 9  1]
 [ 1 28]]
```

In [16]:
```python
TN = confusion[0,0]
TP = confusion[1,1]
FP = confusion[0,1]
FN = confusion[1,0]
```

In [17]:
```python
print('Specificity -- ', round(TN/(FP+TN), 2))
print('Precision -- ', round(TP/(TP+FP), 2))
```

```
Specificity --  0.9
Precision --  0.97
```

# Predict with SVM and dimensionality reduction

#### PCA for dimentionality reduction

In [18]:
```
best_score = 0

# Hyperparameter tuning
for l in range(2,10):
    x_train, x_test, y_train, y_test = tts(X,Y, test_size = 0.20, random_state = 12)
    pca = PCA(n_components=l)
    pc = pca.fit(X)
    scale = preprocessing.StandardScaler().fit(x_train)
    x_train, x_test = scale.transform(x_train), scale.transform(x_test)
    x_train, x_test = pc.transform(x_train), pc.transform(x_test)
    for i in np.arange(start = 0.05, stop = 2.05, step = 0.05):
        for j in np.arange(start = 0.001, stop = 0.101, step = 0.001):
            model = SVC(C = i, gamma = j)
            model.fit(x_train, y_train)
            score = model.score(x_test, y_test)
            if score > best_score:
                best_score = score
                best_C = model.C
                best_gamma = model.gamma
                #best_n_neighbors = iso.n_neighbors
                best_n_components = pca.n_components
print ("The highest score obtained:", round(best_score,2))
print ("C value:", round(best_C,2))
print ("gamma value:", round(best_gamma,2))
print ("pca n_components:", best_n_components)
```

```
The highest score obtained: 0.87
C value: 0.4
gamma value: 0.08
pca n_components: 5
```

In [19]:
```
pred = model.predict(x_test)
print(pd.Series( y_test.value_counts()))
print('Null Accuracy: ',str(round(y_test.value_counts().head(1)[1]/len(y_test) * 100))+'%')
print(f'Test Accuracy: {metrics.accuracy_score(y_test, np.where(pred>=.5, 1, 0)) * 100}')
print(f'Train Accuracy: {metrics.accuracy_score(y_train, np.where(model.predict(x_train)>=.5, 1, 0)) * 100}')

print()
# Confusion matrix
test = y_test
prediction =  np.where(model.predict(x_test)>=.5, 1, 0)
confusion = metrics.confusion_matrix(test, prediction); print(f'Confusion:\n{confusion}')
```

```
1    29
0    10
Name: status, dtype: int64
Null Accuracy:  74.0%
Test Accuracy: 87.17948717948718
Train Accuracy: 94.23076923076923

Confusion:
[[ 5  5]
 [ 0 29]]
```

**Isomap for dimentionality reduction**

In [20]:
```
best_score = 0
for k in range(2, 10):
    for l in range(2, 10):
        x_train, x_test, y_train, y_test = tts(X,Y, test_size = 0.2, random_state = 12)
        iso = Isomap(n_neighbors = k, n_components = l)
        so = iso.fit(X)
        x_train, x_test = so.transform(x_train), so.transform(x_test)
        for i in np.arange(start = 0.05, stop = 2.05, step = 0.05):
            for j in np.arange(start = 0.001, stop = 0.1, step = 0.002):
                model = SVC(C = i, gamma = j)
                model.fit(x_train, y_train)
                score = model.score(x_test, y_test)
                if score > best_score:
                    best_score = score
                    best_C = model.C
                    best_gamma = model.gamma
                    best_n_neighbors = iso.n_neighbors
                    #best_n_components = pca.n_components
print ("The highest score obtained:", round(best_score,2))
print ("C value:", round(best_C,2))
print ("gamma value:", round(best_gamma,2))
print ("pca n_components:", best_n_components)
```

```
The highest score obtained: 0.9
C value: 0.8
gamma value: 0.0
pca n_components: 5
```

In [21]:
```python
pred = model.predict(x_test)
print(pd.Series( y_test.value_counts()))
print('Null Accuracy: ',str(round(y_test.value_counts().head(1)[1]/len(y_test) * 100))+'%')
print(f'Test Accuracy: {metrics.accuracy_score(y_test, np.where(pred>=.5, 1, 0)) * 100}')
print(f'Train Accuracy: {metrics.accuracy_score(y_train, np.where(model.predict(x_train)>=.5, 1, 0)) * 100}')

print()
# Confusion matrix
test = y_test
prediction =  np.where(model.predict(x_test)>=.5, 1, 0)
confusion = metrics.confusion_matrix(test, prediction); print(f'Confusion:\n{confusion}')
```

```
1    29
0    10
Name: status, dtype: int64
Null Accuracy:  74.0%
Test Accuracy: 79.48717948717949
Train Accuracy: 99.35897435897436

Confusion:
[[ 2  8]
 [ 0 29]]
```

In [21]: