```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import pandas as pd
```

```
dataset = pd.read_csv('Data/dataset_spine_clean.csv')
```

```
dataset.columns
```

```
    Index(['Unnamed: 0', 'pelvic_incidence', 'pelvic tilt',
           'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius',
           'degree_spondylolisthesis', 'pelvic_slop', 'Direct_tilt',
           'thoracic_slope', 'cervical_tilt', 'sacrum_angle', 'scoliosis_slope',
           'label', 'label_val'],
          dtype='object')
```

Building our train and test datasets

```
X = pd.DataFrame(
        dataset.drop(axis=1, labels=['label', 'label_val']),
        columns=dataset.columns[:-2])
```

We will just normalize teh datas(the distance between the max and min will be closer).

```
scaler_model = StandardScaler()
X = scaler_model.fit_transform(X)
```

```
y = dataset.label_val
```

```
X
```

```
array([[-1.72647253,  0.14708636,  0.50136873, ...,  1.1671291 ,
        -1.19658726,  1.71236843],
       [-1.71529795, -1.24586434, -0.74876898, ...,  1.67955123,
        -0.94032533, -0.91394145],
       [-1.70412337,  0.4843695 ,  0.46793218, ...,  1.63596949,
        -1.22717809, -0.61568643],
       ...,
       [ 1.70412337,  0.05520137,  0.51561812, ..., -1.05158278,
         1.44337397, -0.69303204],
       [ 1.71529795, -0.88582307, -0.88565951, ..., -0.75264852,
         1.62384854,  0.77376463],
       [ 1.72647253, -1.54892681, -1.24785954, ...,  0.62174631,
         1.29742294,  1.43371339]])
```

```
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
305    1
306    1
307    1
308    1
309    1
Name: label_val, Length: 310, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y, random_state=1)
```

We split our data in $\approx 33\%$ for validation and $\approx 66\%$ for training

```
len(X_test) / len(X_train)
```

```
0.33620689655172414
```

```
def test_model(model):
    plt.plot(model.loss_curve_)
    plt.grid()
    plt.show()

    y_pred = model.predict(X_test)

    confMatrix = metrics.confusion_matrix(y_test, y_pred)
    sns.heatmap(
        pd.DataFrame(confMatrix),
        annot=True
    )
    plt.title("Confusion Matrix")
    plt.ylabel("Right Label")
    plt.xlabel("Predicted Label")

    return {'Score': model.score(X_test, y_test), 'Params': model.get_params()}
```

Base model used for our testing

First Model

- **activation function**: relu
- **solver for weight optimization:** Adam
- **L2 penalty for regularization**: 0.0001
- **Batch_size**: min(200,n_samples)
- **hidden layers sizes**: (100,)
- **Number of layers**: 3
- **learning rate**: constant equal to 0.001
- **epochs** = 30

```
model1 = MLPClassifier(random_state=1, max_iter=30,verbose=True).fit(X_train, y_train);
test_model(model1);
```

In the second model we experimented with the network's topology

Second Model

- **activation function**: relu
- **solver for weight optimization:** Adam
- **L2 penalty for regularization**: 0.0001
- **Batch_size**: min(200,n_samples)

- **hidden layers sizes**: (12,)
- **Number of layers**: 3
- **learning rate**: constant equal to 0.001
- **epochs** = 30

Results

Compared to the previous model, this network performed badly, we think this is because we need larger hidden layers.

```
model2 = MLPClassifier(random_state=1, max_iter=30,verbose=True, hidden_layer_sizes=(12,)).fit(X_train, y_train)
test_model(model2)
```

In this third model we added more hidden layers

Third Model

- **activation function**: relu
- **solver for weight optimization:** Adam
- **L2 penalty for regularization**: 0.0001
- **Batch_size**: min(200,n_samples)
- **hidden layers sizes**: (12,5)
- **Number of layers**: 3

- **learning rate**: constant equal to 0.001
- **epochs** = 30

Results

It's possible to see that increasing the number of hidden layers yielded a better performance for the network. We need more epochs to fully train this network.

```
model3 = MLPClassifier(random_state=1, max_iter=30,verbose=True, hidden_layer_sizes=(12,5)).fit(X_train, y_train)
test_model(model3)
```

In this forth model we changed the learning rate

Forth Model

- **activation function**: relu
- **solver for weight optimization:** Adam
- **L2 penalty for regularization**: 0.0001
- **Batch_size**: min(200,n_samples)
- **hidden layers sizes**: (12, 5)
- **Number of layers**: 3
- **learning rate**: constant equal to 0.2
- **epochs** = 30

Results

It worsened the network's performance, but it converged with the same number of epochs.

```
model4 = MLPClassifier(random_state=1, max_iter=30,verbose=True, hidden_layer_sizes=(12,5), learning_rate_init=0.2
test_model(model4)
```

Here we increased the number of epohcs to fully train this network

Fifth Model

- **activation function**: relu
- **solver for weight optimization:** Adam
- **L2 penalty for regularization**: 0.0001
- **Batch_size**: min(200,n_samples)
- **hidden layers sizes**: (200,)
- **Number of layers**: 3
- **learning rate**: constant equal to 0.001
- **epochs** = 300

Results

The training eventually converged and it shows a good loss.

```
model5 = MLPClassifier(random_state=1, max_iter=300,verbose=True, hidden_layer_sizes=(12,5)).fit(X_train, y_train)
test_model(model5)
```