Documentation Part 1 – Πρόβλεψη, Ταξινόμηση, Ομαδοποίηση σε Καρδιοτοκογραφικά Δεδομένα

Στοιχεία:

Π18105, Γεώργιος Μπαλτζάκης

Requirements:

- jupyter lab/notebook.
- Python 3.9
- Sklearn
- Keras
- Pandas
- Numpy
- Matplotlib
- mpl_toolkits

Σημείωση:

- Ο κώδικας που θα ακολουθήσει βρίσκεται στο φάκελο:
  Code/Part1.ipynb
- Τα δεδομένα(data) βρίσκονται στο φάκελο:
  Data/data.csv
- Τα δεδομένα που περιέχει το data.csv είναι οι 23 στήλες(21 χαρακτηριστικά + Original Classification(*Class*) + Original Clustering(*NSP*)) από το αρχικό αρχείο στην ενότητα *Data.*

Επεξήγηση Κώδικα

Ερώτημα 1 (Data Cleaning):

```python
# Question 1 - Data Cleaning #

# Convert float format from '0,1' to '0.1'. Also in this data set all numpy.object types are floats
for i in data.columns:
    if(data[i].dtype == np.object):
        for j in range(data[i].size):
            tmp = []
            tmp = data[i][j].split(",")
            if (tmp != []):
                try:
                    data[i][j] = np.float64(tmp[0] + '.' + tmp[1])
                except:
                    pass
        data[i] = pd.to_numeric(data[i])

# Drop negative values except column -> 'Tendency'
for i in data.columns[:-3]:
     data = data.drop(data[data['LB'] < 0].index)

# Drop if heart bpm < 60
data = data.drop(data[data['LB'] < 60].index)

# Drop if heart bpm > 200
data = data.drop(data[data['LB'] > 200].index)

# Drop duplicate rows
data = data.drop_duplicates()

# Drop rows with NaN values
data = data.dropna()
```

Σε αυτό το μέρος ο κώδικας καθαρίζει προετοιμάζει τα δεδομένα από:

- μη λογικές τιμές
- κενές εγγραφές
- διπλές εγγραφές

Ερώτημα 2 (Clustering):

Υλοποίηση *Clustering* με:

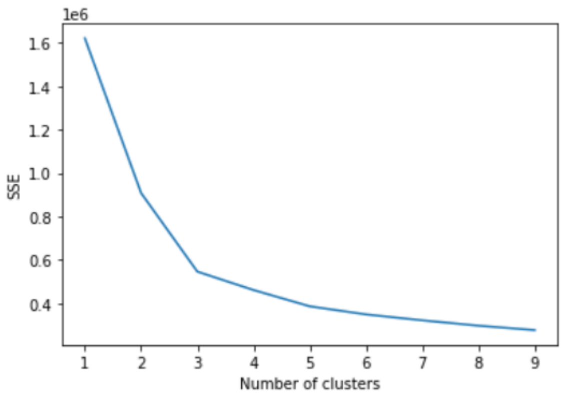
- K-Means
- DBSCAN
- OPTICS

Βήμα 1ο - Προετοιμασία για Clustering:

- Επιλέχθηκαν τα 11 πρώτα χαρακτηριστικά καθώς τα υπόλοιπα αφορούν Μεταβλητές του Ιστογράμματος.
- Εύρεση κατάλληλου πλήθους *Συστάδων(Clusters)* - υλοποιήθηκε με τις μεθόδους:
    1. *SSE*(Sum of Squared Errors)
    2. *Elbow method.*
       (πηγή)
- *Κανονικοποίηση* - υλοποιήθηκε με τη βιβλιοθήκη:
    1. *sklearn.preprocessing.StandardScaler*

```
# Question 2 - Clustering

# Data preparation for Clustering
data_clustering = data[data.columns[:-12]]
data_clustering.head()
print("\n-----------------------------------------------------------------\n")
data_clustering.info()

# 1) Find the best number of clusters (SSE / Elbow Method)
print("\n-----------------------------------------------------------------\n")
print("Find the best number of clusters with SSE & Elbow Method.\n")
sse = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k).fit(data_clustering)
    data_clustering["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_ # Inertia: Sum of distances of samples to their closest cluster center
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of clusters")
plt.ylabel("SSE")
plt.show()

print("\nSo, the best number of clusters is 3.\n\nK = 3")
print("\n-----------------------------------------------------------------\n")

original_labels = data['NSP'].copy()

# 2) Standard Scaler
data_clustering = StandardScaler().fit_transform(data_clustering)
```

Έξοδος:




```
So, the best number of clusters is 3.

K = 3
```

Βήμα 2ο - Υλοποίηση K-Means:

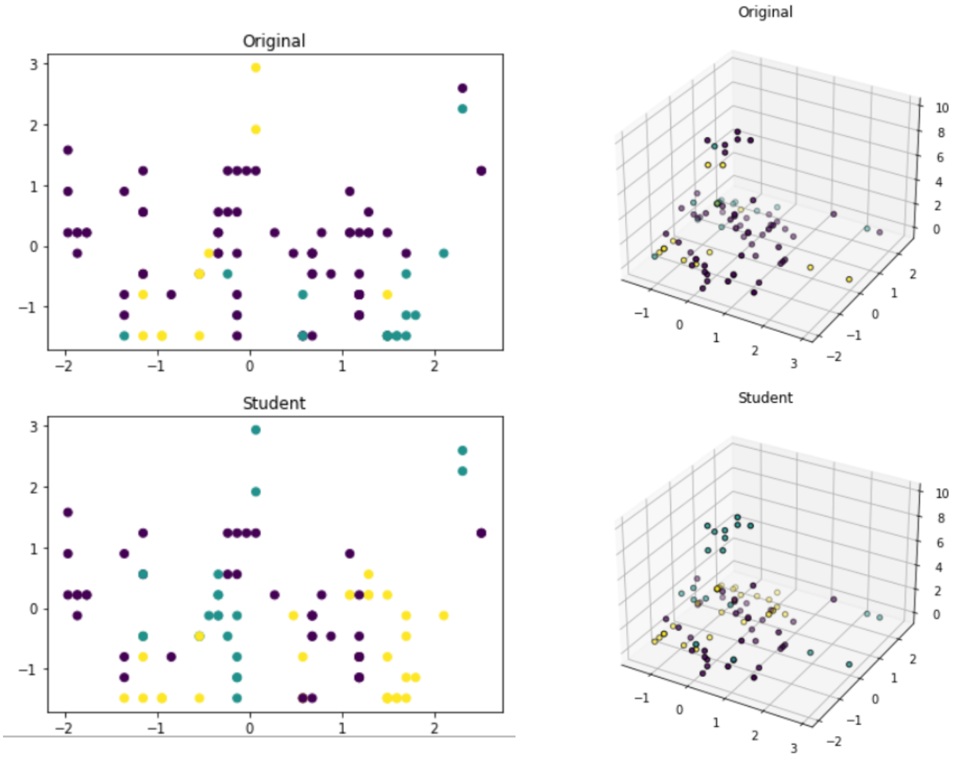
<u>Σημείωση:</u> Την μέθοδο visualization θα την εξηγήσουμε αργότερα.

```python
# K-Means #

# Student
kmeans = KMeans(n_clusters=3, random_state=0)
kmeans = kmeans.fit(data_clustering)
student_labels = kmeans.labels_
```

```python
# Visualization
visualization("K-Means", original_labels, data_clustering, student_labels)
```

Έξοδος:

K-Means Visualization

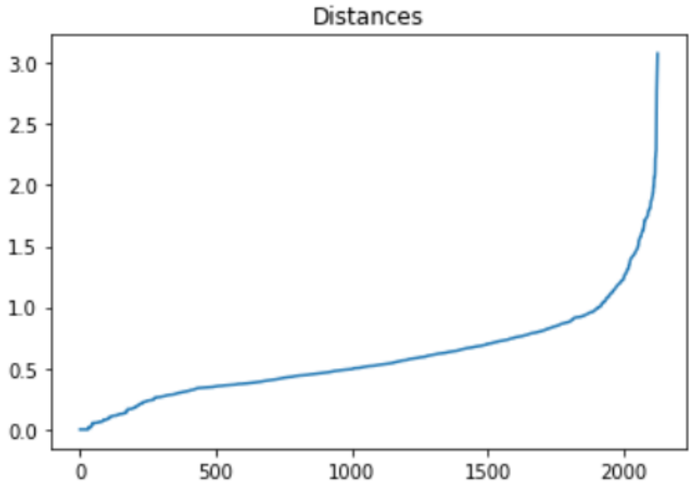


Βήμα 3ο – Προετοιμασία για DBSCAN:

- Εύρεση *e*(epsilon) με τις μεθόδους:
    1. *NearestNeighbors(),* από τη βιβλιοθήκη:
       sklearn.neighbors.NearestNeighbors
    2. *Elbow method'*
       (πηγή)

```
# Find the best eps (Elbow method)
neigh = NearestNeighbors(n_neighbors = 11)
nbrs = neigh.fit(data_clustering)
distances, indices = nbrs.kneighbors(data_clustering)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
plt.title("Distances")
plt.show()
print("\nEPS = 1.0")
```

Έξοδος:




```
EPS = 1.0
```

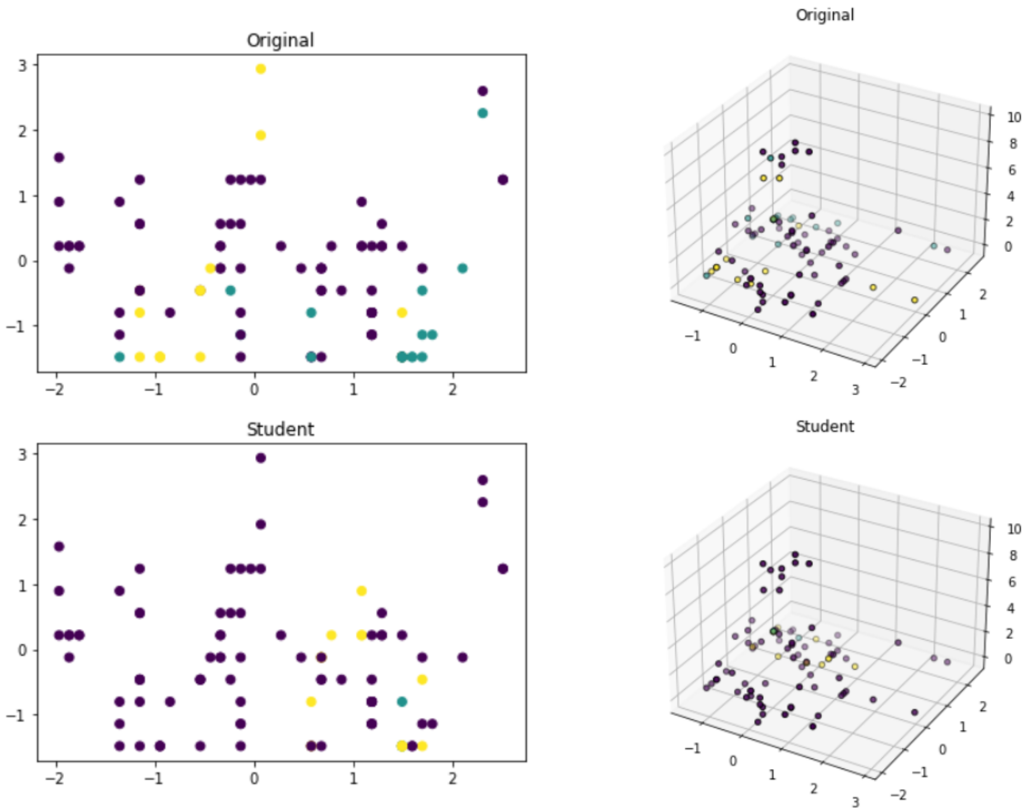
Βήμα 4ο – Υλοποίηση DBSCAN:

```
# DBSCAN #

# Student
dbscan = DBSCAN(eps = 1.0, min_samples=2 * 11)
dbscan = dbscan.fit(data_clustering)
student_labels = dbscan.labels_
```

```
# Visualization
visualization("DBSCAN", original_labels, data_clustering, student_labels)
```

Έξοδος:

```
DBSCAN Visualization
```



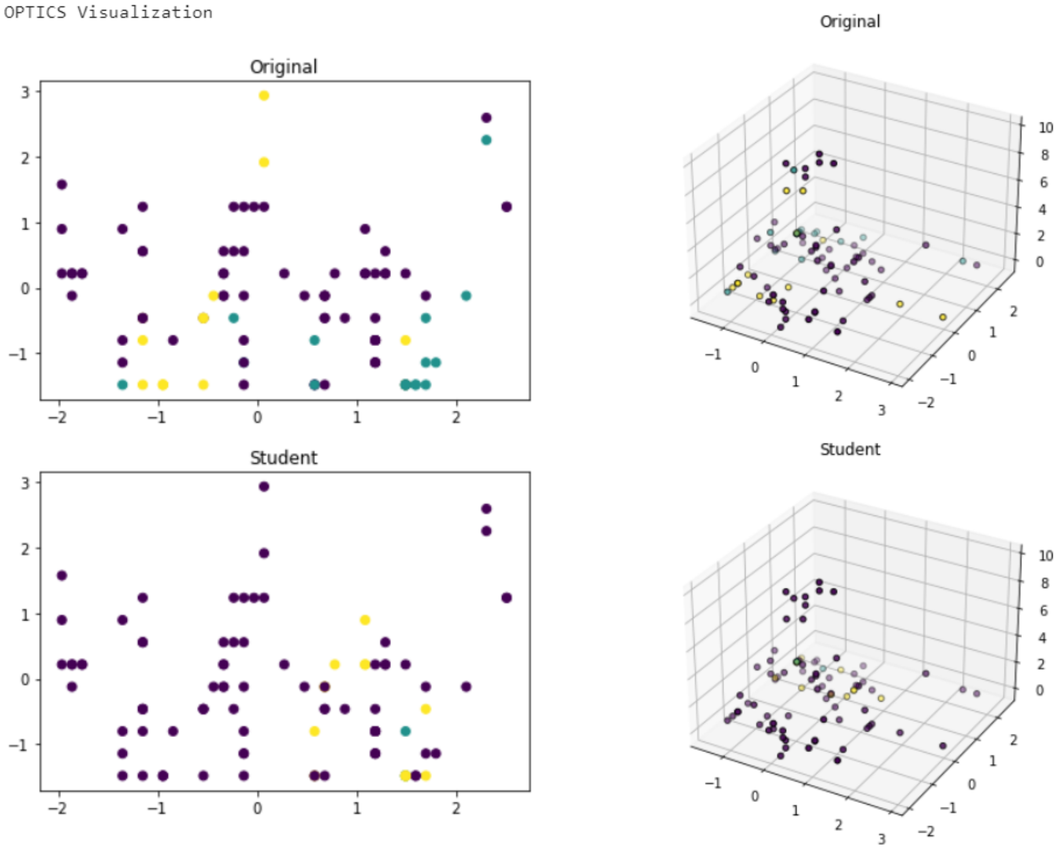
Βήμα 5ο – Υλοποίηση OPTICS

```
# Optics

# Student
optics = OPTICS(min_samples=5, max_eps=80) # define the model
optics = optics.fit(data_clustering) # fit the model
labels = optics.labels_
```

```
# Visualization
visualization("OPTICS", original_labels, data_clustering, student_labels)
```

Έξοδος:

```
OPTICS Visualization
```



Ερώτημα 3 (Classification):

Βήμα 1ο – Προετοιμασία δεδομένων:

- Για το Classification χρειαζόμαστε:
    1. Τα 21 χαρακτηριστικά, και
    2. Τη στήλη CLASS έτσι ώστε να κάνουμε την αξιολόγηση της εκπαίδευσης
- Κανονικοποίηση
- Θέτουμε τα δεδομένα:
    1. Εκπαίδευσης 60%
    2. Αξιολόγησης 10%
    3. Ελέγχου 30%

```
# Question 3 - Classification #

# Initializing data_classification
data_classification = data[data.columns[:-2]]
fhr = data[data.columns[-2]]
data_classification = StandardScaler().fit_transform(data_classification)
```

```
# Preparation of data #
X = data_classification.astype(float)
Y = fhr

# Convert data to dummy variables
encoder = LabelEncoder().fit(Y)
y_bool = encoder.transform(Y)
y = np_utils.to_categorical(y_bool)

# Split data into train and test
len_data = X.shape[0]
train_size = int(len_data * .6)
test_size = int(len_data * .3)
valid_size = int(len_data * .1)
print ("Train size: %d" % train_size)
print ("Test size: %d" % test_size)
print ("Validation size: %d" % valid_size)

xtr = X[:train_size,:]
ytr = y[:train_size,:]
ytr_bool = y_bool[:train_size]

xva = X[train_size:train_size+valid_size,:]
yva = y[train_size:train_size+valid_size,:]
yva_bool = y_bool[train_size:train_size+valid_size]

xte = X[train_size+valid_size:,:]
yte = y[train_size+valid_size:,:]
yte_bool = y_bool[train_size+valid_size:]
```

Βήμα 2ο – Δημιουργία Νευρωνικού Δικτύου:

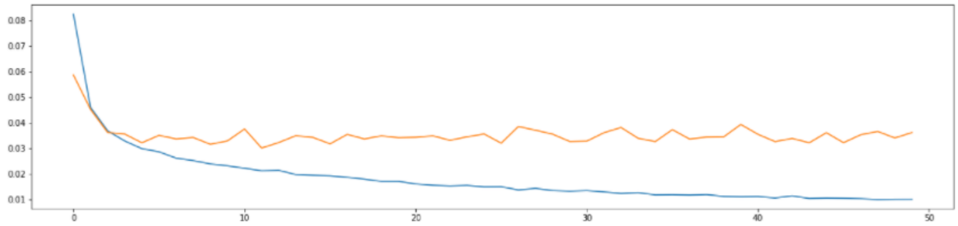
Χαρακτηριστικά Νευρωνικού Δικτύου 1:

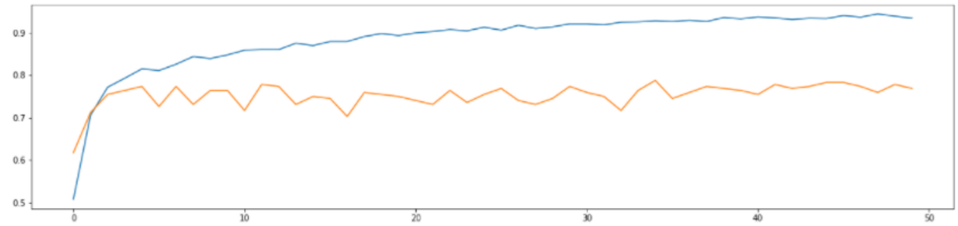
- 4 Στρώματα Νευρώνων(1-Εισόδου, 2-Κρυφά, 1-Εξόδου)
    1. Input → 21 χαρακτηριστικά εισόδου
    2. 1ο Κρυφό Στρώμα → 350 Νευρώνες
    3. 2ο Κρυφό Στρώμα → 50 Νευρώνες
    4. Output → 10 Κατηγορίες
- Εποχές → 50
- Batch size → 10

```
# Define the Model v1
model = Sequential()
model.add(Dense(350, input_dim = 21, activation = 'relu'))
model.add(Dense(50, activation = 'relu'))
model.add(Dense(10, activation = 'sigmoid'))

# Compile the Model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# from keras.utils import plot_model
# plot_model(model, to_file='model.png')

# Fit model
history = model.fit(xtr, ytr,
                    validation_data=(xva, yva),
                    epochs=50,
                    batch_size=10,
                    verbose=0)

# Plot training and validation loss
plt.figure(figsize=(20, 10))
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.show()

# Evaluate and Predict
scores = model.evaluate(xtr, ytr, verbose=0)
print("Train %s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

ytr_pred = model.predict_classes(xtr, verbose=0)
print("Train Accuracy by model.predict: %.2f%%" % (100*sum(ytr_bool == ytr_pred)/ytr.shape[0]))

# make class predictions with the model
yva_pred = model.predict_classes(xva, verbose=0)
print("Val Accuracy by model.predict: %.2f%%" % (100*sum(yva_bool == yva_pred)/yva.shape[0]))

# make class predictions with the model
yte_pred = model.predict(xte, batch_size=1, verbose=0)
yte_pred_bool = np.argmax(yte_pred, axis=1)

print("Test Accuracy by model.predict: %.2f%%" % (100*sum(yte_bool == yte_pred_bool)/yte.shape[0]))

# Matthews correlation coefficient
print("Train MMC: ", matthews_corrcoef(ytr_bool, ytr_pred))
print("Val MMC: ", matthews_corrcoef(yva_bool, yva_pred))
print("Test MMC: ", matthews_corrcoef(yte_bool, yte_pred_bool))

print("--------------TRAIN--------------------")
print(confusion_matrix(ytr_bool, ytr_pred))
print("--------------VAL--------------------")
print(confusion_matrix(yva_bool, yva_pred))
print("--------------TEST--------------------")
print(confusion_matrix(yte_bool, yte_pred_bool))
```

Έξοδος:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_16 (Dense)             (None, 350)               7700      
_________________________________________________________________
dense_17 (Dense)             (None, 50)                17550     
_________________________________________________________________
dense_18 (Dense)             (None, 10)                510       
=================================================================
Total params: 25,760
Trainable params: 25,760
Non-trainable params: 0
_________________________________________________________________
None
```





```
Train accuracy: 93.96%
c:\users\georg\appdata\local\programs\python\python39\lib\site-packages\keras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
Train Accuracy by model.predict: 93.96%
Val Accuracy by model.predict: 76.89%
Test Accuracy by model.predict: 56.81%
Train MMC:  0.9280082791084835
Val MMC:  0.7059896648598883
Test MMC:  0.5117390727572328
---------------TRAIN--------------------
[[259   2   1   0   0   1   1   0   0   4]
 [ 11 333   0   1   0   1   0   0   0   0]
 [  8   0  31   0   0   0   0   0   0   0]
 [  0   0   0  71   0   0   0   0   0   0]
 [  4   0   0   0  36   0   0   0   0   4]
 [  2   1   0   0   0 121   1   0   0   0]
 [  0   0   0   0   0   0 113   0   0   0]
 [  0   0   1   0   0   0  11   0   1   0]
 [  0   0   0   0   0   0   0   0  50  19]
 [  3   0   0   0   0   0   0   0   0 184]]
---------------VAL--------------------
[[17  0  0  1  0  0  0  0  0]
 [ 4 54  6  2  1  0  0  1  0]
 [ 0  0  3  0  0  0  0  0  0]
 [ 7  2  0  9  0  0  0  0  0]
 [ 0  6  0  0 51  4  0  0  0]
 [ 0  0  0  0  9 25  0  0  0]
 [ 0  0  0  0  0  4  0  0  0]
 [ 0  0  0  0  0  0  0  0  0]
 [ 0  0  0  2  0  0  0  0  4]]
---------------TEST--------------------
[[ 19   5   1   0   0   0   2   0  13  58]
 [  3 121   0  26   2   2   2   0   4   5]
 [  8   0   5   0   0   0   0   0   0   1]
 [  0   2   0   5   0   0   0   0   0   0]
 [  4   0   0   0   6   0   0   0   0   0]
 [  1   8   0   9   0 100  26   0   2   0]
 [  1   0   0   0   0   1 103   0   0   0]
 [  0   0   0   0   0   5  55   0  30   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   4]]
```

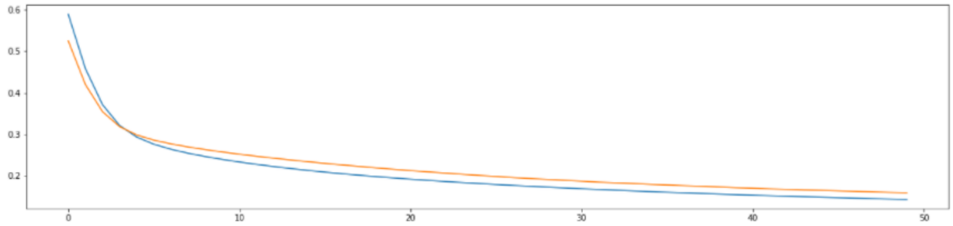
Χαρακτηριστικά Νευρωνικού Δικτύου 2:

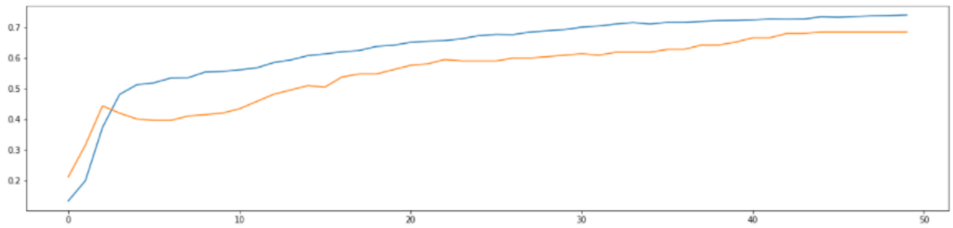
- *4 Στρώματα Νευρώνων(1-Εισόδου, 2-Κρυφά, 1-Εξόδου)*
    1. *Input → 21 χαρακτηριστικά εισόδου*
    2. *1ο Κρυφό Στρώμα → 500 Νευρώνες*
    3. *2ο Κρυφό Στρώμα → 100 Νευρώνες*
    4. *Output → 10 Κατηγορίες*
- *Εποχές → 50*
- *Batch size → 10*

```python
# Define the Model v2
model = Sequential()
model.add(Dense(500, input_dim = 21, activation = 'relu'))
model.add(Dense(100, activation = 'relu'))
model.add(Dense(10, activation = 'sigmoid'))

# Compile the Model
model.compile(loss='binary_crossentropy', optimizer='sgd', metrics=['accuracy'])

print(model.summary())

# from keras.utils import plot_model
# plot_model(model, to_file='model.png')

# Fit model
history = model.fit(xtr, ytr,
                    validation_data=(xva, yva),
                    epochs=50,
                    batch_size=10,
                    verbose=0)

# Plot training and validation loss
plt.figure(figsize=(20, 10))
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.show()

# Evaluate and Predict
scores = model.evaluate(xtr, ytr, verbose=0)
print("Train %s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

ytr_pred = model.predict_classes(xtr, verbose=0)
print("Train Accuracy by model.predict: %.2f%%" % (100*sum(ytr_bool == ytr_pred)/ytr.shape[0]))

# make class predictions with the model
yva_pred = model.predict_classes(xva, verbose=0)
print("Val Accuracy by model.predict: %.2f%%" % (100*sum(yva_bool == yva_pred)/yva.shape[0]))

# make class predictions with the model
yte_pred = model.predict(xte, batch_size=1, verbose=0)
yte_pred_bool = np.argmax(yte_pred, axis=1)

print("Test Accuracy by model.predict: %.2f%%" % (100*sum(yte_bool == yte_pred_bool)/yte.shape[0]))

# Matthews correlation coefficient
print("Train MMC: ", matthews_corrcoef(ytr_bool, ytr_pred))
print("Val MMC: ", matthews_corrcoef(yva_bool, yva_pred))
print("Test MMC: ", matthews_corrcoef(yte_bool, yte_pred_bool))

print("---------------TRAIN--------------------")
print(confusion_matrix(ytr_bool, ytr_pred))
print("---------------VAL--------------------")
print(confusion_matrix(yva_bool, yva_pred))
print("---------------TEST--------------------")
print(confusion_matrix(yte_bool, yte_pred_bool))
```

Έξοδος:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 500)               11000
_________________________________________________________________
dense_20 (Dense)             (None, 100)               50100
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1010
=================================================================
Total params: 62,110
Trainable params: 62,110
Non-trainable params: 0
_________________________________________________________________
None
```





```
Train accuracy: 74.35%
c:\users\georg\appdata\local\programs\python\python39\lib\site-packages\keras\engine\sequential.py:450: U
serWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use inst
ead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if
it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model
does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
Train Accuracy by model.predict: 74.35%
Val Accuracy by model.predict: 68.40%
Test Accuracy by model.predict: 48.36%
Train MMC:  0.6890356070985061
Val MMC:  0.6085232940546534
Test MMC:  0.41449088963720904
---------------TRAIN---------------------
[[218  26   0   0   0   0   1   0   0  23]
 [ 20 316   0   4   0   5   0   0   0   1]
 [ 30   9   0   0   0   0   0   0   0   0]
 [  0  22   0  49   0   0   0   0   0   0]
 [ 22   6   0   1   0   0   0   0   0  15]
 [  1  29   0   1   0  78  16   0   0   0]
 [ 12   0   0   0   0   4  96   0   0   1]
 [  0   0   0   0   0   1   5   7   0   0]
 [  4   0   0   0   0   0   0   0  29  36]
 [ 21   3   0   0   0   0   1   0   7 155]]
---------------VAL---------------------
[[15  1  0  0  0  0  0  2]
 [ 5 57  3  0  0  0  0  3]
 [ 0  2  1  0  0  0  0  0]
 [13  1  1  1  0  0  0  2]
 [ 0  7  0  0 40 14  0  0]
 [ 2  0  0  0  3 29  0  0]
 [ 0  0  0  0  0  4  0  0]
 [ 4  0  0  0  0  0  0  2]]
---------------TEST---------------------
[[ 41   2   0   0   0   0   5   0  17  33]
 [ 16 127   0  16   0   0   2   0   0   4]
 [ 12   2   0   0   0   0   0   0   0   0]
 [  0   4   0   3   0   0   0   0   0   0]
 [  4   2   0   0   0   0   0   0   0   4]
 [  1  19   0  46   0  29  50   0   0   1]
 [  4   0   0   0   0   1 100   0   0   0]
 [  0   1   0   0   0   0  84   5   0   0]
 [  0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   4]]
```