

ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής



Μάθημα: «ΑΝΑΛΥΤΙΚΗ ΔΕΔΟΜΕΝΩΝ (60 εξ.)»

Π18101 – ΑΝΑΣΤΑΣΙΑ ΙΩΑΝΝΑ ΜΕΞΑ

Π18123 – ΒΑΣΙΛΙΚΗ ΠΑΣΙΑ

Π18164 – ΠΕΤΡΟΣ ΕΥΣΤΑΘΙΟΣ ΦΑΤΟΥΡΟΣ

Σας έχουμε αποστείλει τους κώδικες και σε .py και σε .ipynb, σε περίπτωση που τα αρχεία του Google Colab δεν σας δουλεύουν για οποιονδήποτε λόγο.

ΜΕΡΟΣ Α:

Ερώτημα 1: Προπαρασκευή δεδομένων – υλοποίηση με εργαλεία στατιστικής επεξεργασίας.

Τα δεδομένα που χρησιμοποιήσαμε για την εκπόνηση της εργασίας, βρίσκονται στο Excel αρχείο που κατεβάζουμε από την ιστοσελίδα που αναγράφεται στην εκφώνηση και συγκεκριμένα στο φύλλο εργασίας “Data”. Από εκεί διαλέξαμε όλα τα χαρακτηριστικά που είναι αριθμημένα στο αρχείο με μπλε αριθμούς. Για λόγους ευκολίας, όλα τα χαρακτηριστικά που χρησιμοποιήθηκαν αναγράφονται στον παρακάτω πίνακα.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|-----|-----|------|--------|------|------|--------|----------|----------|-------|------|-------|
| LB | AC | FM | UC | DL | DS | DP | ASTV | MSTV | ALTV | MLTV | Width |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | |
| Min | Max | Nmax | Nzeros | Mode | Mean | Median | Variance | Tendency | CLASS | NSP | |

Χρησιμοποιώντας το Excel ως εργαλείο, προχωρήσαμε στην διαδικασία προπαρασκευής των δεδομένων. Δεν διαγράψαμε τίποτα από το dataset, διότι δεν υπήρχε κάποια κενή εγγραφή, ούτε θεωρήσαμε ότι υπήρχαν εσφαλμένες τιμές. Συνεχίσαμε στην κανονικοποίηση των δεδομένων, καθώς και στην διακριτοποίησή τους, ωστόσο δεν χρησιμοποιήσαμε τα διακριτοποιημένα δεδομένα, διότι οι αλγόριθμοι clustering που χρησιμοποιούμε σε επόμενο ερώτημα, δίνουν ως ονόματα στις συστάδες που δημιουργούν αριθμητικές τιμές, όπως και το MLP λειτουργεί με αριθμητικές τιμές. Παρακάτω αναλύονται οι διαδικασίες τις κανονικοποίησης και διακριτοποίησης:

- Κανονικοποίηση, είναι η μετατροπή αριθμητικών τιμών σε άλλες, πιο «κατάλληλες», αριθμητικές τιμές.
- Διακριτοποίηση, είναι ο μετασχηματισμός των αριθμητικών τιμών σε αλφαριθμητικές τιμές.

Αποφασίσαμε να μετατρέψουμε όλες τις τιμές του dataset σε τιμές που να κυμαίνονται στην περιοχή [0,1]. Αυτό το επιτύγχαμε με την χρήση του Excel ως εργαλείο. Αρχικά, σε κάθε στήλη που θέλαμε να κανονικοποιήσουμε, βρήκαμε το μέγιστο και το ελάχιστο στοιχείο και ο τύπος που χρησιμοποιήσαμε για να δημιουργήσουμε την νέα κανονικοποιημένη στήλη, είναι:

$$x' = \frac{x - min}{max - min}$$

- x' είναι η κανονικοποιημένη τιμή
- x είναι η αρχική τιμή
- min είναι η τιμή του ελάχιστου στοιχείου
- max είναι η τιμή του μέγιστου στοιχείου

| 1 | LB normalized |
|-----|---------------|
| 120 | 0,259 |
| 132 | 0,481 |
| 133 | 0,500 |
| 134 | 0,519 |
| 132 | 0,481 |
| 134 | 0,519 |
| 140 | 0,630 |
| 140 | 0,630 |
| 140 | 0,630 |
| 140 | 0,630 |
| 140 | 0,630 |
| 142 | 0,667 |

| min | 106 | 0 |
|-----|-----|---|
| max | 160 | 1 |

Στην διπλανή εικόνα διακρίνεται ένα παράδειγμα κανονικοποίησης της στήλης LB. Για λόγους απεικόνισης δεν φαίνονται όλα τα δεδομένα, μόνο μερικά στην αρχή και στο τέλος του dataset.

Πραγματοποιήσαμε διακριτοποίηση μόνο για λόγους πληρότητας, στις στήλες που αναγράφονται στον παρακάτω πίνακα.

| 21 | 22 | 23 |
|----------|-------|-----|
| Tendency | CLASS | NSP |

Αντιστοιχήσαμε τις αριθμητικές τιμές με τις κατάλληλες αλφαριθμητικές, όπως επεξηγούνται στο φύλλο εργασίας "Description" του dataset. Ακολουθεί screenshot που φαίνονται οι συναρτήσεις που χρησιμοποιήθηκαν και τα αποτελέσματα τις διακριτοποίησης.

Tendency =IFS(AS3 = -1; "left asymmetric"; AS3 = 0; "symmetric"; AS3 = 1; "right asymmetric")

CLASS =IFS(BG3 = 1; "A"; BG3 = 2; "B"; BG3 = 3; "C"; BG3 = 4; "D"; BG3 = 5; "SH"; BG3 = 6; "AD"; BG3 = 7; "DE"; BG3 = 8; "LD"; BG3 = 9; "FS"; BG3 = 10; "SUSP")

NSP =IFS(BI3 = 1; "Normal"; BI3 = 2; "Suspect"; BI3 = 3; "Pathologic")

| 21 | 22 | 23 | | | |
|----------|-------------------------|-------|----------------------|-----|--------------------|
| Tendency | Tendency discretization | CLASS | CLASS discretization | NSP | NSP discretization |
| 1 | right asymmetric | 9 | FS | 2 | Suspect |
| 0 | symmetric | 6 | AD | 1 | Normal |
| 0 | symmetric | 6 | AD | 1 | Normal |
| 1 | right asymmetric | 6 | AD | 1 | Normal |
| 1 | right asymmetric | 2 | B | 1 | Normal |
| 0 | symmetric | 8 | LD | 3 | Pathologic |
| 0 | symmetric | 1 | A | 1 | Normal |
| 0 | symmetric | 5 | SH | 2 | Suspect |
| 1 | right asymmetric | 5 | SH | 2 | Suspect |
| 1 | right asymmetric | 5 | SH | 2 | Suspect |
| 1 | right asymmetric | 5 | SH | 2 | Suspect |
| 0 | symmetric | 1 | A | 1 | Normal |

Μόλις ολοκληρώσαμε τις διαδικασίες που περιγράφθηκαν παραπάνω, συνεχίσαμε στην δημιουργία ενός νέου αρχείου Excel με όνομα «CTG(1).xlsx», το οποίο σας το έχουμε παραδώσει, που περιλαμβάνει όλα τα δεδομένα στην κατάλληλη μορφή τους, που θα χρησιμοποιήσουμε στα παρακάτω ερωτήματα.

Ερώτημα 2: clustering – υλοποίηση με scikit-learn.

Έχουμε υλοποιήσει τρεις αλγορίθμους clustering, το K-means, το DBSCAN και το OPTICS. Ακόμα κάνουμε την χρήση κάποιων δεικτών για να αξιολογήσουμε τα αποτελέσματα των αλγορίθμων. Οι δείκτες που χρησιμοποιήσαμε εξηγούνται παρακάτω:

- Homogeneity score: ο δείκτης αυτός μετράει κατά πόσο μια συστάδα αποτελείται από δείγματα που ανήκουν μόνο σε μια κλάση. Οι τιμές του κυμαίνονται από 0 μέχρι 1.
- Completeness score: σκοπός αυτού του δείκτη, είναι να παρέχει πληροφορία σχετικά με την ανάθεση των δειγμάτων που ανήκουν στην ίδια κλάση. Πιο συγκεκριμένα, ένας καλός clustering αλγόριθμος, αντιστοιχεί όλα τα δείγματα της ίδιας κλάσης στην ίδια συστάδα. Οι τιμές του κυμαίνονται από 0 μέχρι 1.
- V-measure: για τον υπολογισμό αυτού του δείκτη, είναι απαραίτητο να έχουν υπολογιστεί οι 2 προηγούμενοι. Στην ουσία ο v-measure αποτελεί τον αρμονικό μέσο του homogeneity score και του completeness score. Οι τιμές του κυμαίνονται από 0 μέχρι 1.

Μετά την χρήση του κάθε αλγορίθμου, δημιουργούμε και τις confusion matrices ανάμεσα στα πραγματικά αποτελέσματα και στις συστάδες που έχουν δημιουργηθεί από τους αλγορίθμους. Στον διπλανό πίνακα C διαστάσεων 2x2, το κελί C_{00} είναι ο αριθμός των true negatives, το κελί C_{10} είναι ο αριθμός των false negatives, στο C_{11} είναι ο αριθμός των true positives και στο C_{01} είναι ο αριθμός των false positives.

| | |
|----------|----------|
| C_{00} | C_{01} |
| C_{10} | C_{11} |

Επεξήγηση κώδικα

Το αρχείο κώδικα που υλοποιούνται οι τρεις αλγόριθμοι clustering, ονομάζεται «clusters.ipynb». Χρησιμοποιήσαμε το Google Colab όπως μας δείξατε στα εργαστήρια του μαθήματος.

Αρχικά, κάνουμε import όλες τις κατάλληλες βιβλιοθήκες και εργαλεία που θα χρησιμοποιήσουμε στην συνέχεια του κώδικα.

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.cluster import OPTICS
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from sklearn.metrics.cluster import homogeneity_score
from sklearn.metrics.cluster import v_measure_score
from sklearn.metrics.cluster import completeness_score
from sklearn.metrics.cluster import pair_confusion_matrix
```

Σημείωση: Αν η τελευταία import σας βγάζει error, παρακαλούμε να τρέξετε το επόμενο μπλοκ κώδικα που περιέχει μια εντολή pip install, η οποία αναβαθμίζει την έκδοση του scikit-learn και μόλις ολοκληρωθεί θα σας ζητηθεί να κάνετε restart το session. Έπειτα, μπορείτε κανονικά να τρέξετε όλα τα import. Αν το μπλοκ κώδικα με τα import τρέξει κανονικά και δεν βγάλει κάποιο πρόβλημα τότε δεν χρειάζεται να τρέξετε και την εντολή pip install.

 # If the last import doesn't work, please run the code below
!pip install --upgrade scikit-learn

Στην συνέχεια, φορτώνουμε τα δεδομένα μας από το Excel αρχείο «CTG(1).xlsx», το οποίο είναι απαραίτητο να το φορτώσετε στο περιβάλλον του Google Colab, για να τρέξει κανονικά ο κώδικας. Φτιάχνουμε 2 λίστες με τα ονόματα input και output που περιέχουν τα δεδομένα εισόδου και σε ποια κλάση έχουν καταταχθεί τα δεδομένα, αντίστοιχα. Τέλος, δημιουργούμε ένα γράφημα δύο διαστάσεων με τα χαρακτηριστικά LB και ASTV. Πρέπει να σημειωθεί ότι στο παρακάτω screenshot ως δεδομένα κατάταξης έχουμε πάρει την στήλη V του Excel αρχείου δηλαδή το χαρακτηριστικό CLASS, παρόλα αυτά θα τρέξουμε παραδείγματα και με τα δύο χαρακτηριστικά – στόχους (FHR, NSP).



Ξεκινάμε να χτίσουμε το μοντέλο για τον αλγόριθμο K-means, μέσω του εργαλείου scikit-learn. Αρχικά, ορίζουμε το μοντέλο με παραμέτρους `n_clusters = 10`, διότι έχουμε 10 διαφορετικές κλάσεις και `random_state = 0`, για να παίρνουμε σε κάθε εκτέλεση του αλγορίθμου τα ίδια αποτελέσματα (αν το `None` που είναι το default, κάθε φορά θα παίρναμε διαφορετικά αποτελέσματα). Στην συνέχεια εκπαιδεύουμε το μοντέλο μας με την εντολή `.fit(input)` δίνοντας τα δεδομένα εισόδου και μας επιστρέφονται τα αποτελέσματα στην μεταβλητή `labels` (οι εκτιμήσεις του αλγορίθμου). Τέλος, για χάρη της καλύτερης προβολής των αποτελεσμάτων, δημιουργούμε ένα γράφημα δύο διαστάσεων με τα χαρακτηριστικά LB και ASTV, τυπώνονται τα αποτελέσματα των δεικτών και το confusion matrix, καθώς και το πόσα clusters και με τι ονομασία δημιουργήθηκαν.

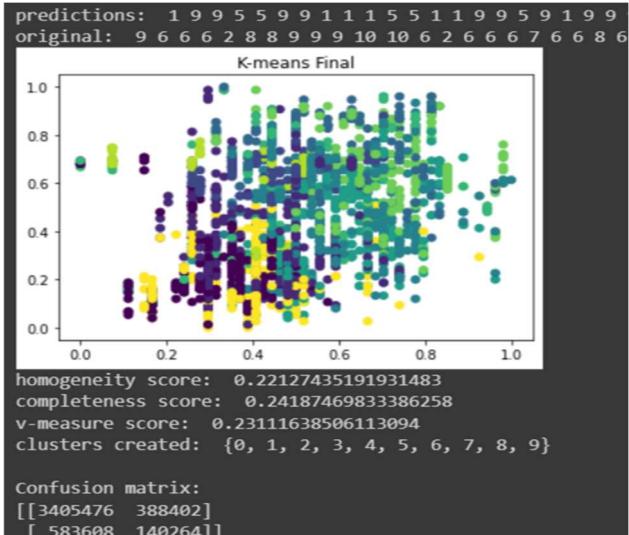
```
# Using the K-means cluster
kmeans = KMeans(n_clusters=10, random_state=0) # define the model
kmeans = kmeans.fit(input) # fit the model
labels = kmeans.labels_ # predictions
print("\npredictions: ", *labels)
print("original: ", *output.flatten().tolist())

# plot the results
plt.scatter(input[:,0], input[:,7], c=labels)
plt.title("K-means Final")
plt.show()

# Metrics
print("homogeneity score: ", homogeneity_score(labels.flatten().tolist(), output.flatten().tolist()))
print("completeness score: ", completeness_score(labels.flatten().tolist(), output.flatten().tolist()))
print("v-measure score: ", v_measure_score([labels.flatten().tolist(), output.flatten().tolist()])

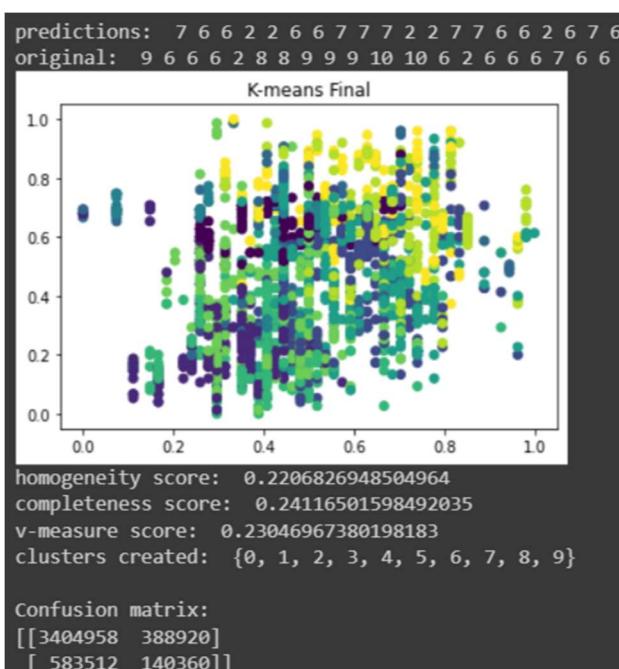
# printing the unique values of the list
print("clusters created: ", set(labels))

# Confusion matrix
print("\nconfusion matrix: ")
print(pair_confusion_matrix(output.flatten().tolist(),labels.tolist()))
```

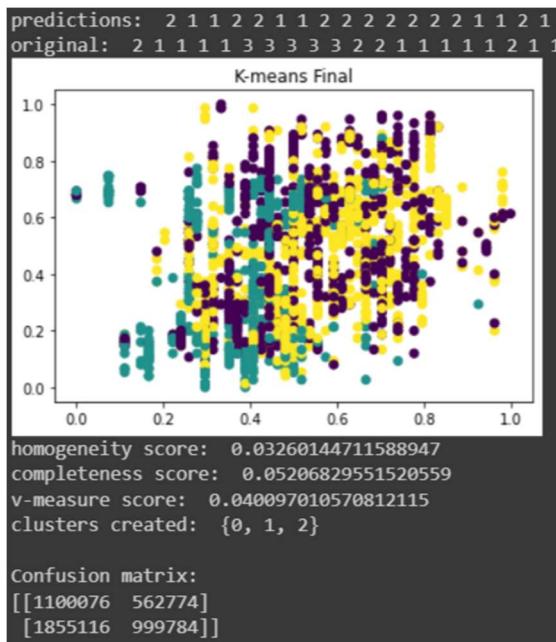


Προσπαθήσαμε να δώσουμε τιμή και σε άλλες παραμέτρους, ωστόσο ότι παράδειγμα και αν δοκιμάσαμε, τα αποτελέσματα στα metrics και στο confusion matrix δεν είχαν καμία διαφορά. Παρακάτω, φαίνεται ένα screenshot των αποτελεσμάτων, τρέχοντας το μοντέλο με έξτρα παραμέτρους το `init = "random"`, δηλαδή η αρχικοποίηση των centroids θα γίνει με τυχαίο τρόπο και `max_iter = "300"`, δηλαδή ο μέγιστος αριθμός επαναλήψεων που θα τρέξει ο αλγόριθμος.

```
kmeans = KMeans(n_clusters=10, init="random", max_iter=300, random_state=0) # define the model
```



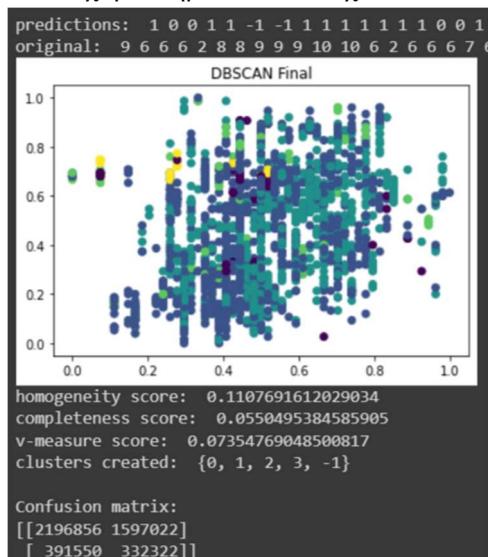
Συνεχίσαμε και τρέξαμε τον αλγόριθμο και για το χαρακτηριστικό – στόχο NSP. Τα μόνα σημεία που πρέπει να αλλάξουν στον κώδικα είναι, στην αρχή που ορίζουμε την λίστα output, να δέχεται τα δεδομένα από την στήλη W του Excel αρχείου και ο αριθμός των clusters πρέπει να είναι ίσος με 3 ($n_{clusters} = 3$).



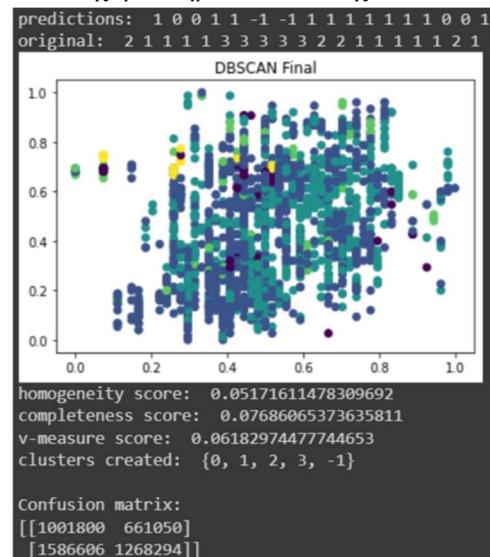
Παρατηρούμε ότι τα αποτελέσματα στα metrics είναι αρκετά χειρότερα όταν ως χαρακτηριστικά – στόχο έχουμε μόνο 3 κλάσεις, από όταν είχαμε 10 κλάσεις. Στην περίπτωση των 10 κλάσεων (FHR), τα αποτελέσματα των metrics κυμαίνονταν γύρω στο 0.2, που και πάλι δεν θεωρείται καλό σκορ σε σχέση με το άριστα που είναι 1, ωστόσο στην περίπτωση των 3 κλάσεων (NSP) τα αποτελέσματα των metrics είναι πιο απογοητευτικά και λαμβάνουν κάποια τιμή στο δεύτερο δεκαδικό ψηφίο. Αυτό είναι λογικό, γιατί στην μια περίπτωση δημιουργούμε για τον ίδιο αριθμό δεδομένων 10 συστάδες, ενώ στην άλλη δημιουργούμε 3 συστάδες, επομένως υπάρχουν περισσότερα περιθώρια για λάθη.

Το επόμενο κομμάτι κώδικα, υλοποιεί τον αλγόριθμο DBSCAN. Δεν διαφέρει σε τίποτα από το κομμάτι κώδικα του K-means, παρά μόνο στο πως χτίζουμε το μοντέλο για το DBSCAN. Για τον ορισμό του μοντέλου, δίνουμε ως παραμέτρους $\text{eps} = 0.5$, δηλαδή ποια θα είναι η μέγιστη απόσταση μεταξύ δύο σημείων ώστε το ένα να θεωρείται γείτονας του άλλου και $\text{min_samples} = 15$, δηλαδή το πλήθος των σημείων μιας περιοχής, ώστε ένα σημείο να θεωρηθεί το κεντρικό σημείο. Για τον υπολογισμό της απόστασης, χρησιμοποιούμε την default επιλογή, δηλαδή την ευκλείδεια απόσταση. Χρησιμοποιώντας τις παραπάνω τιμές στις παραμέτρους, φαίνεται να παίρνουμε τα καλύτερα δυνατά αποτελέσματα, τα οποία απεικονίζονται στο παρακάτω screenshot.

Για χαρακτηριστικό – στόχο το FHR



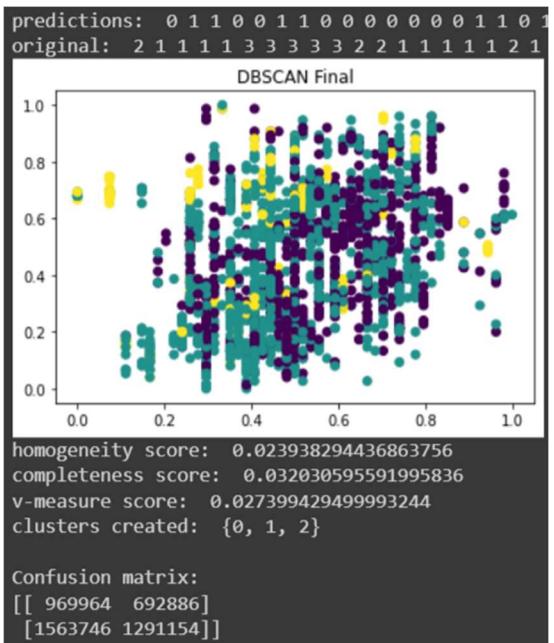
Για χαρακτηριστικό – στόχο το NSP



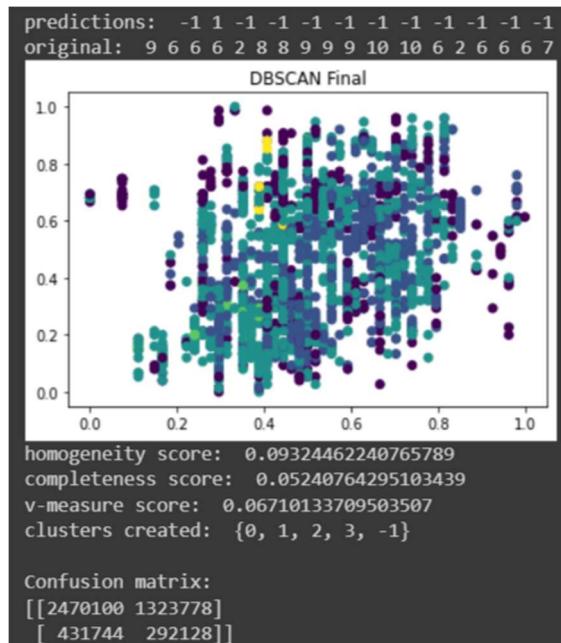
Τα αποτελέσματα των metrics πάλι δεν είναι ενθαρρυντικά, ωστόσο και αυτό εξηγείται, διότι και στις δύο περιπτώσεις που έχουμε 10 κλάσεις και 3 κλάσεις, δημιουργούνται 5 συστάδες. Επομένως δεν υπάρχει καλή ακρίβεια σε καμία περίπτωση, γιατί στην μια περίπτωση περνάμε από πολλές κλάσεις (10) σε λιγότερες συστάδες (5) και στην άλλη περίπτωση, περνάμε από λίγες κλάσεις (3) σε περισσότερες συστάδες (5).

Αλλάζοντας τις τιμές των παραμέτρων δεν φαίνεται να βελτιώνει την κατάσταση, ίσα ίσα βγαίνουν χειρότερα αποτελέσματα. Παραθέτουμε μερικά screenshots:

eps = 1, min_samples = 5 (NSP)

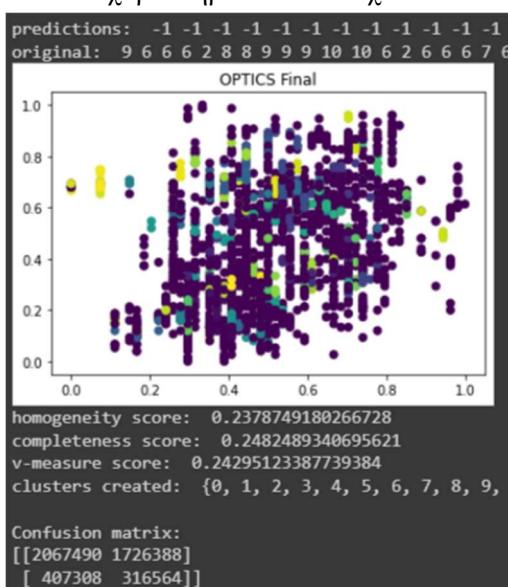


eps = 0.3, min_samples = 10 (FHR)

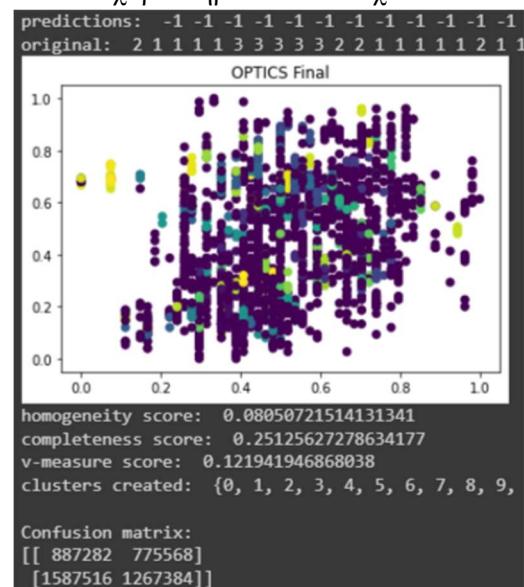


Το επόμενο κομμάτι κώδικα, υλοποιεί τον αλγόριθμο OPTICS. Δεν διαφέρει σε τίποτα από το κομμάτι κώδικα του K-means και του DBSCAN, παρά μόνο στο πως χτίζουμε το μοντέλο για το OPTICS. Για τον ορισμό του μοντέλου, δίνουμε ως παραμέτρους max_eps = 0.5, δηλαδή ποια θα είναι η μέγιστη απόσταση μεταξύ δύο σημείων ώστε το ένα να θεωρείται γείτονας του άλλου και min_samples = 5, δηλαδή το πλήθος των σημείων μιας περιοχής, ώστε ένα σημείο να θεωρηθεί το κεντρικό σημείο. Για τον υπολογισμό της απόστασης, χρησιμοποιούμε την default επιλογή, δηλαδή την απόσταση minkowski. Χρησιμοποιώντας τις παραπάνω τιμές στις παραμέτρους, φαίνεται να πάρνουμε τα καλύτερα δυνατά αποτελέσματα, τα οποία απεικονίζονται στο παρακάτω screenshot.

Για χαρακτηριστικό – στόχο το FHR



Για χαρακτηριστικό – στόχο το NSP



Τα αποτελέσματα των metrics μοιάζουν πολύ με αυτά του K-means όσον αφορά το χαρακτηριστικό – στόχο FHR. Βέβαια εδώ έχουν δημιουργηθεί παραπάνω από 10 συστάδες. Για το χαρακτηριστικό – στόχο NSP, τα αποτελέσματα των metrics είναι καλύτερα από αυτά του K-means.

Και για τα δύο χαρακτηριστικά – στόχους, τα καλύτερα αποτελέσματα λαμβάνονται μέσω του OPTICS. Για το FHR, αμέσως καλύτερη προσέγγιση δίνει ο K-means και η χειρότερη δίνει ο DBSCAN. Ενώ για το NSP, αμέσως καλύτερη απόδοση δίνει ο DBSCAN και χειρότερη δίνει ο K-means.

Ερώτημα 3: classification – υλοποίηση με keras/tensorflow.

Έχουμε υλοποιήσει ένα MLP νευρωνικό δίκτυο με την χρήση του εργαλείου keras/tensorflow. Χωρίσαμε τα δεδομένα μας σε train set, validation set και test set όπως μας λέει η εκφώνηση, εκπαιδεύσαμε το νευρωνικό δίκτυο και το χρησιμοποιήσαμε για να προβλέψουμε το χαρακτηριστικό - στόχο FHR και τέλος εκτυπώνονται τα accuracy για κάθε set δεδομένων, καθώς και οι confusion matrices.

Σημείωση: Δεν έγινε χρήση του Matthews correlation coefficient, διότι είναι ένα μέτρο σύγκρισης μόνο για binary classifications (two-class).

Επεξήγηση κώδικα

Το αρχείο κώδικα που υλοποιείται το νευρωνικό δίκτυο MLP ονομάζεται «MLP.ipynb». Χρησιμοποιήσαμε το Google Colab όπως μας δείξατε στα εργαστήρια του μαθήματος.

Αρχικά, κάνουμε import όλες τις κατάλληλες βιβλιοθήκες και εργαλεία που θα χρησιμοποιήσουμε στην συνέχεια του κώδικα.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
import tensorflow as tf
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
import matplotlib.pyplot as plt
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix
```

Στην συνέχεια, φορτώνουμε τα δεδομένα μας από το Excel αρχείο «CTG(1).xlsx», το οποίο είναι απαραίτητο να το φορτώσετε στο περιβάλλον του Google Colab, για να τρέξει κανονικά ο κώδικας. Φτιάχνουμε 2 λίστες με τα ονόματα input και output που περιέχουν τα δεδομένα εισόδου και σε ποια κλάση έχουν καταταχθεί τα δεδομένα, αντίστοιχα με βάση το χαρακτηριστικό – στόχο FHR. Επίσης, μετατρέπουμε τα δεδομένα εξόδου σε μορφή one hot vector encoding για την καλύτερη λειτουργία του MLP.

```
X = pd.read_excel('CTG(1).xlsx',usecols = "A:U") # loading the input data
Y = pd.read_excel('CTG(1).xlsx',usecols = "V") # loading the output data
input = X[1:].values
output = Y[1:].values

# Converting the output data using the one hot vector encoding
output_en = np_utils.to_categorical(output)

# Examples of conversion
print(output[:5])
print(output_en[:5, :])
```

```
[[9]
 [6]
 [6]
 [6]
 [2]]
[[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

Χωρίζουμε τα δεδομένα εισόδου και εξόδου σε training set που καλύπτει το 60% των δεδομένων, σε validation set που καλύπτει το 10% και σε testing set που καλύπτει το 30%.

```
# Split data into train and test
len_data = input.shape[0]
print(len_data)
train_size = int(len_data * .6)
valid_size = int(len_data * .1)
print ("Train size (60 %): ", train_size)
print ("Validation size (10 %): ", valid_size)
print ("Test size (30 %): ", (len_data - (train_size+valid_size)))

xtr = input[:train_size,:]
ytr = output_en[:train_size,:]
ytr_bool = output[:train_size]

xva = input[train_size:train_size+valid_size,:]
yva = output_en[train_size:train_size+valid_size,:]
yva_bool = output[train_size:train_size+valid_size]

xte = input[train_size+valid_size:,:]
yte = output_en[train_size+valid_size:,:]
yte_bool = output[train_size+valid_size:]
```

```
2126
Train size (60 %): 1275
Validation size (10 %): 212
Test size (30 %): 639
```

Έπειτα, ορίζουμε το ακολουθιακό μοντέλο μας και ορίζουμε τον αριθμό των νευρώνων που θα υπάρχουν στο input layer ίσο με 21, αφού έχουμε 21 χαρακτηριστικά, στα δύο hidden layers αποφασίσαμε να δώσουμε ως αριθμό νευρώνων το 25. Παρατηρήσαμε ότι φτάνουμε σε κοντινό μέγεθος accuracy με το να δίναμε 100 νευρώνες στα hidden layers, ωστόσο η διαφορά ήταν ότι όταν είχαμε 100 νευρώνες, το μοντέλο μας φαινόταν να υπερ-εκπαιδεύεται και για αυτό αποφασίσαμε να κρατήσουμε ως πλήθος νευρώνων το 25. Θα αναλυθούν όλες οι περιπτώσεις που δοκιμάσαμε σχετικά με το πλήθος των νευρώνων παρακάτω. Συνεχίζοντας, ως activation function στα ενδιάμεσα στρώματα δίνουμε την «relu» και στο output layer έχουμε 11 νευρώνες και ως activation function δίνουμε την «softmax», διότι έχουμε ένα πρόβλημα classification.

Παρατήρηση: Έχουμε 11 νευρώνες στο output layer, διότι έχουμε μετατρέψει τα δεδομένα εξόδου σε μορφή one hot vector encoding που μας επέστρεψε αποτελέσματα μήκους 11.

Έπειτα, πρέπει να κάνουμε compile το μοντέλο μας και χρησιμοποιούμε ως loss function την categorical crossentropy, αφού έχουμε ένα πρόβλημα πολλών κλάσεων, optimizer χρησιμοποιούμε τον adam που βοηθάει στην ελαχιστοποίηση της loss function και τέλος ως metric χρησιμοποιούμε το accuracy.

```
# Define model
model = Sequential()
model.add(Dense(25, input_shape = (21,), activation='relu'))
model.add(Dense(25, activation='relu'))
#model.add(Dropout(0.5))
model.add(Dense(11, activation='softmax'))

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

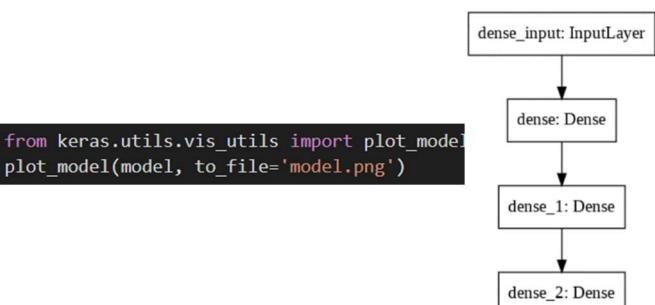
print(model.summary())
```

| Model: "sequential" | | |
|---------------------|--------------|---------|
| Layer (type) | Output Shape | Param # |
| dense (Dense) | (None, 25) | 550 |
| dense_1 (Dense) | (None, 25) | 650 |
| dense_2 (Dense) | (None, 11) | 286 |

Total params: 1,486
Trainable params: 1,486
Non-trainable params: 0
None

Για λόγους πληρότητας, φτιάχνουμε και σχηματικά την μορφή του μοντέλου.

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

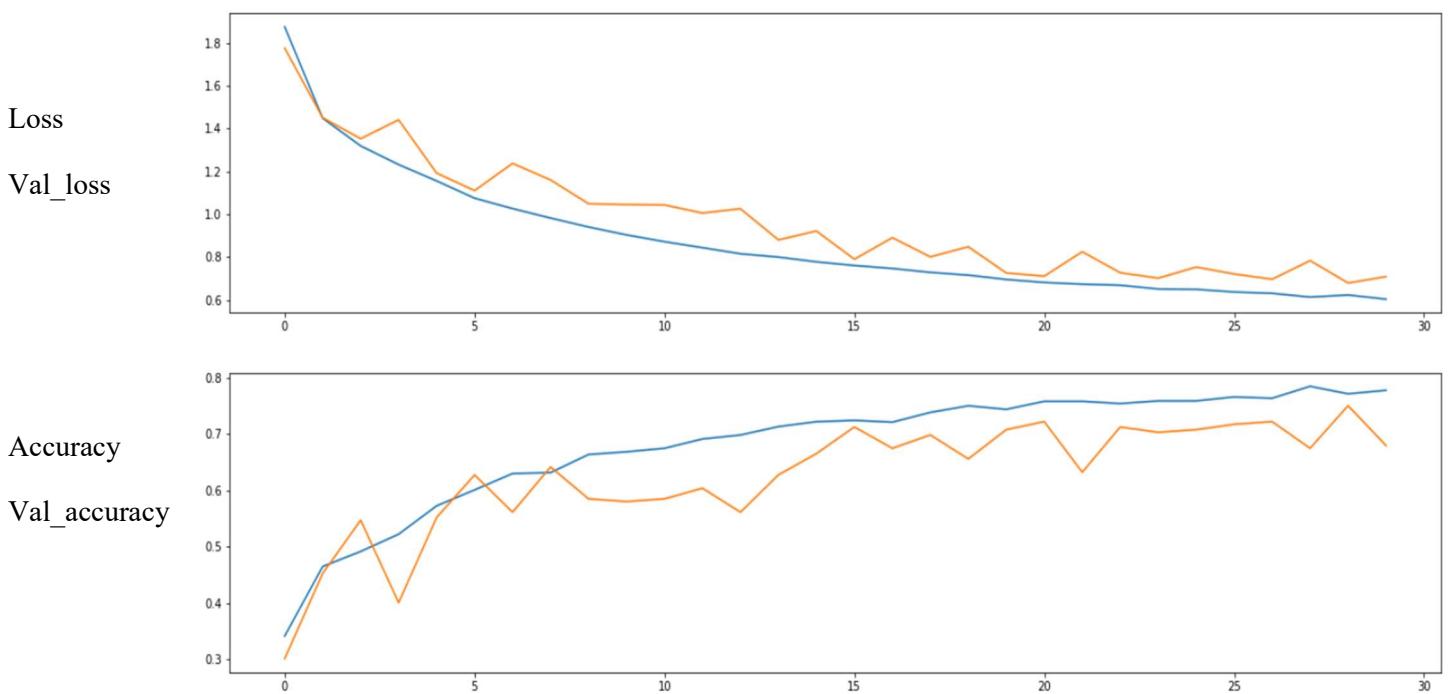


Ηρθε η ώρα να εκπαιδεύσουμε το μοντέλο. Με την χρήση του keras αυτό γίνεται μόλις με μία εντολή, την .fit(), όπου ως παραμέτρους δίνουμε τα training data, τα testing data και τα validation data, αριθμό epoch έχουμε ορίσει το 30 το θεωρούμε αρκετό χωρίς να υπερ-εκπαιδεύεται το μοντέλο, batch size ίσο με 1 ώστε κάθε φορά να παίρνει ένα ένα τα δεδομένα και verbose ίσο με 2 για να εκτυπώνονται όλες η πληροφορίες σε κάθε epoch.

```
# Fit model
history = model.fit(tf.convert_to_tensor(xtr, dtype=tf.float32), \
                     ytr, validation_data=(tf.convert_to_tensor(xva, dtype=tf.float32), yva), epochs=30, batch_size=1, verbose=2)
```

Μετά την εκπαίδευση του μοντέλου, δημιουργούμε δύο γραφικές παραστάσεις συγκρίνοντας τα metrics loss και val_loss και σε άλλη γραφική παράσταση συγκρίνουμε τα accuracy και val_accuracy.

```
# 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()
```



Παρατηρούμε ότι, αναμενόμενα οι τιμές της loss function πέφτουν όσο περνάνε τα epochs και οι τιμές του accuracy αυξάνονται. Άρα, το μοντέλο μας φαίνεται να εκπαιδεύεται σωστά. Ωστόσο, για μεγαλύτερη κατανόηση και απεικόνιση των αποτελεσμάτων τα εκτυπώνουμε και στις παρακάτω μορφές, καθώς και τα confusion matrices για κάθε set δεδομένων που δημιουργήσαμε.

```
# Evaluate and Predict
scores = model.evaluate(tf.convert_to_tensor(xtr, dtype=tf.float32), ytr, verbose=0)
print("Train %s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

ytr_pred = model.predict_classes(tf.convert_to_tensor(xtr, dtype=tf.float32), verbose=0)
print("Train Accuracy by model.predict: %.2f%%" % (100*sum(ytr_bool.flatten() == ytr_pred)/ytr.shape[0]))

# make class predictions with the model
yva_pred = model.predict_classes(tf.convert_to_tensor(xva, dtype=tf.float32), verbose=0)
print("Val Accuracy by model.predict: %.2f%%" % (100*sum(yva_bool.flatten() == yva_pred)/yva.shape[0]))

# make class predictions with the model
yte_pred = model.predict(tf.convert_to_tensor(xte, dtype=tf.float32), 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.flatten() == yte_pred_bool)/yte.shape[0]))

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

```

Train accuracy: 79.84%
Train Accuracy by model.predict: 79.84%
Val Accuracy by model.predict: 69.34%
Test Accuracy by model.predict: 54.77%
-----TRAIN-----
[[228  2 18  0  0  0  2  0  0 18]
 [ 53 275  8  0  2  5  1  0  0  2]
 [ 15  0 24  0  0  0  0  0  0  0]
 [  0 16  0 55  0  0  0  0  0  0]
 [ 13  4  0  0 13  0  0  0  0 14]
 [ 10  8  3  0  0 83 20  0  0  1]
 [  5  0  1  0  0  1 105  0  0  1]
 [  0  0  0  0  0  4  9  0  0  0]
 [  0  0  0  0  0  0  0 47 22]
 [  7  1  0  0  0  0  0  0  0 179]]

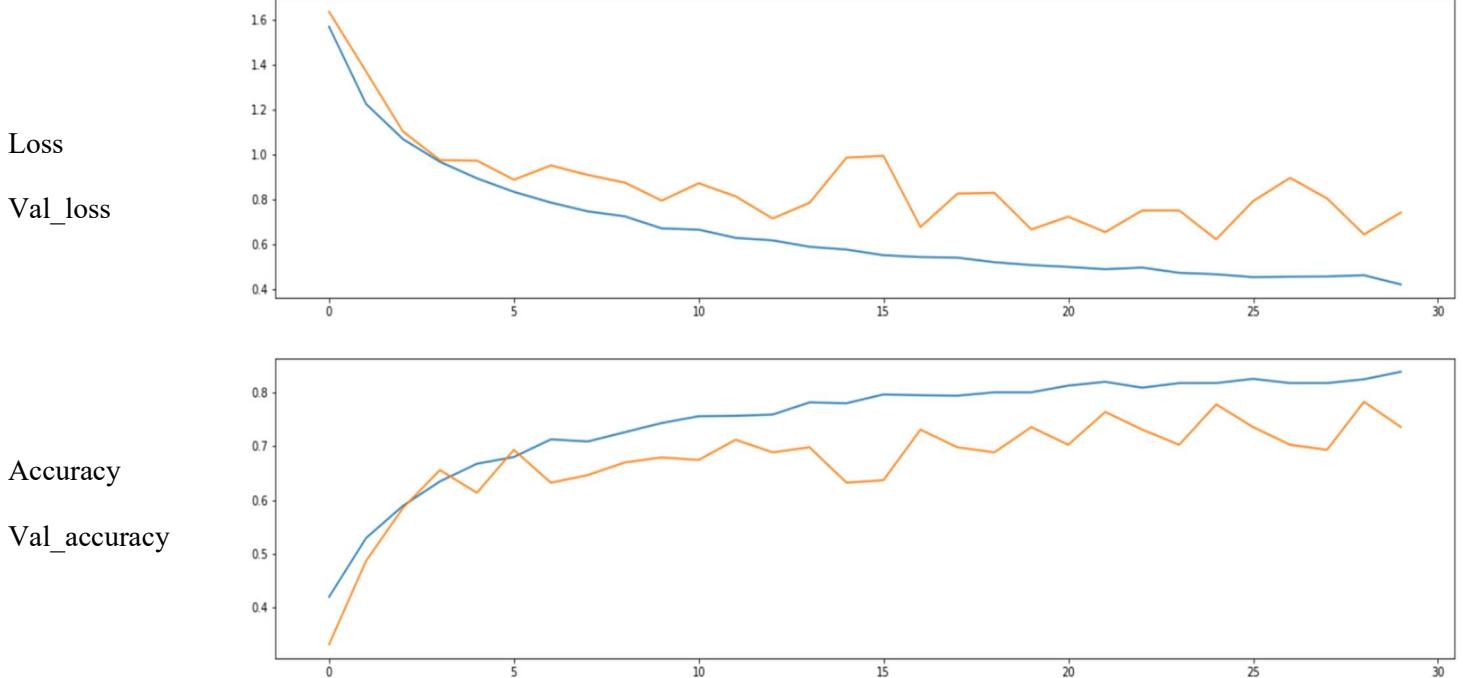
```

| VAL | | | | | | | | | |
|-----|-----|----|---|---|---|----|----|---|-----|
| [| [12 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 5] |
| [| [15 | 48 | 1 | 3 | 0 | 1 | 0 | 0 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [| [0 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0] |
| [| [12 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 2] |
| [| [2 | 3 | 0 | 0 | 0 | 45 | 11 | 0 | 0] |
| [| [2 | 0 | 0 | 0 | 0 | 2 | 30 | 0 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6]] |

| TEST | | | | | | | | | | |
|------|------|-----|---|----|---|----|----|-----|---|-----|
| [| [23 | 11 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 61] |
| [| [14 | 131 | 1 | 9 | 3 | 0 | 2 | 0 | 0 | 5] |
| [| [7 | 0 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1] |
| [| [0 | 5 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0] |
| [| [4 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4] |
| [| [1 | 32 | 0 | 18 | 0 | 31 | 55 | 8 | 0 | 1] |
| [| [1 | 0 | 0 | 0 | 0 | 0 | 0 | 104 | 0 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 2 | 41 | 49 | 0 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| [| [0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4]] |

Συμπερασματικά, βλέπουμε ότι το μοντέλο με πλήθος νευρώνων στα hidden layers to 25 και με 30 epochs κατάφερε να αποσπάσει train accuracy κοντά στο 80%, validation accuracy γύρω στο 70% και test accuracy κοντά στο 55%.

Ας συγκρίνουμε τα παραπάνω αποτελέσματα, τρέχοντας τώρα το μοντέλο με τον ίδιο αριθμό epochs (30), αλλά αντί για 25 νευρώνες στα hidden layers θα βάλουμε 100.



Παρατηρούμε ότι, το accuracy πάλι αυξάνεται και το loss μειώνεται όσο αυξάνονται τα epochs, ωστόσο υπάρχει μια σημαντική απόκλιση μεταξύ των γραφικών παραστάσεων του loss με το val_loss και του accuracy με το val_accuracy. Αυτό σημαίνει ότι το μοντέλο μας έχει την τάση να υπερ-εκπαιδεύεται (overfitting). Ας δούμε συγκριτικά τώρα και τα αποτελέσματα των metrics.

Συγκριτικά με το προηγούμενο μοντέλο, βλέπουμε πως στο train και validation accuracy υπάρχει μια αύξηση περίπου 4%, ωστόσο στο test accuracy παρατηρούμε μια μείωση της τάξεως του 2%.

Επομένως, καταλήγουμε στο συμπέρασμα ότι αυξάνοντας τόσο πολύ των αριθμών των νευρώνων στα hidden layers ναι μεν έχουμε μια μικρή αύξηση στην ακρίβεια, ωστόσο φαίνεται να δημιουργείται το πρόβλημα του overfitting. Αυτό θα προσπαθήσουμε να το διορθώσουμε, κάνοντας χρήση της εντολής dropout() όταν χτίζουμε το μοντέλο. Αυτή είναι μια τεχνική κατά την οποία αγνοούνται τυχαία νευρώνες κατά τη διάρκεια της εκπαίδευσης, με αποτέλεσμα το δίκτυο να είναι πιο ικανό για καλύτερη γενίκευση και είναι λιγότερο πιθανό να υπερβάλουν τα δεδομένα εκπαίδευσης.

Η πιθανότητα που θα ορίσουμε για να συμβεί το dropout, είναι 50%.

```

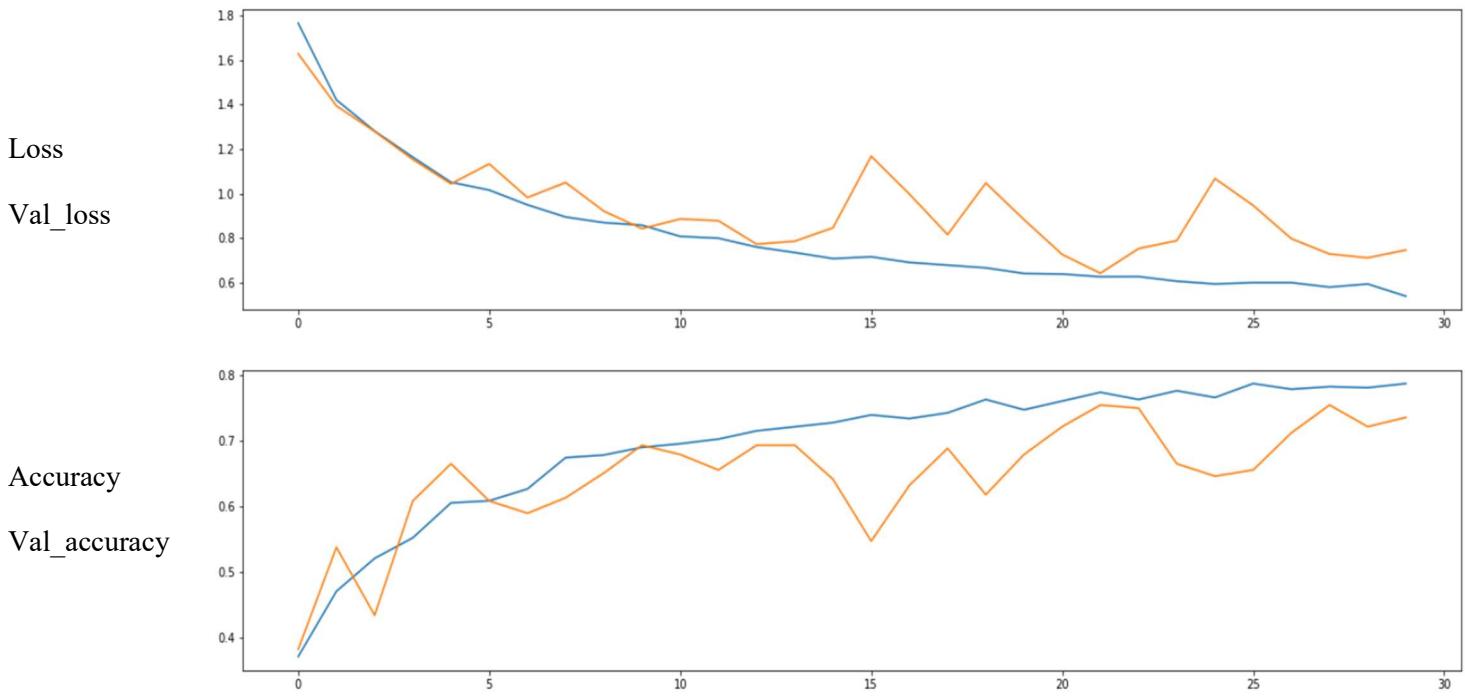
Train accuracy: 83.61%
Train Accuracy by model.predict: 83.61%
Val Accuracy by model.predict: 73.58%
Test Accuracy by model.predict: 52.90%

```

```

# Define model
model = Sequential()
model.add(Dense(100, input_shape = (21,), activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(11, activation='softmax'))

```



Όπως δείχνουν τα αποτελέσματα των metrics, δεν έχουμε σχεδόν καμία μεταβολή στα ποσοστά και οι γραφικές παραστάσεις φαίνεται να έχουν έρθει πιο «κοντά» η μία στην άλλη. Άρα, φαίνεται να υπάρχει μια σχετική βελτίωση στο θέμα του overfitting, ωστόσο υπάρχουν ακόμα διακριτές «διακυμάνσεις».

Train accuracy: 83.61%
Train Accuracy by model.predict: 83.61%
Val Accuracy by model.predict: 73.58%
Test Accuracy by model.predict: 51.49%

ΜΕΡΟΣ Β:

Ερώτημα 1: Προπαρασκευή δεδομένων – υλοποίηση με εργαλεία στατιστικής επεξεργασίας, π.χ. R/Python.

Τα δεδομένα που χρησιμοποιήσαμε για την εκπόνηση της εργασίας, βρίσκονται στο Excel αρχείο που κατεβάζουμε από την ιστοσελίδα που αναγράφεται στην εκφώνηση. Χρησιμοποιώντας το Excel ως εργαλείο, κρατήσαμε μόνο τις πρώτες 530 εγγραφές και διαγράψαμε την στήλη date, όπως μας συμβουλεύεται στην εκφώνηση. Για να προχωρήσουμε στην υλοποίηση της μετατροπής της χρονοσειράς σε supervised πρόβλημα, δημιουργήσαμε καινούργιο αρχείο Excel, το οποίο και σας έχουμε στείλει, με όνομα «data_akbilgic(1).xlsx». Δεν διαφέρει σε τίποτα με το αρχικό αρχείο, εκτός από τα ονόματα των στηλών που τις μετονομάσαμε όλες σε data. Η σειρά με την οποία είναι γραμμένα τα χαρακτηριστικά στο Excel παραμένει η ίδια. Για λόγους διευκόλυνσης, στον παρακάτω πίνακα αναγράφονται τα ονόματα των στηλών και τα αντίστοιχα ονόματα των χαρακτηριστικών:

| A | B | C | D | E | F | G | H | I |
|-----------------|------------------|----|-----|------|--------|---------|----|----|
| ISE TL BASED | ISE USD BASED | SP | DAX | FTSE | NIKKEI | BOVESPA | EU | EM |

Επεξήγηση κώδικα

Τα αρχεία κώδικα που υλοποιούν την συνάρτηση μετατροπής, ονομάζονται «time_series_MLP.ipynb» και «time_series_RNN.ipynb». Χρησιμοποιήσαμε το Google Colab όπως μας δείξατε στα εργαστήρια του μαθήματος.

Αρχικά, κάνουμε import όλες τις κατάλληλες βιβλιοθήκες και εργαλεία που θα χρησιμοποιήσουμε στην συνέχεια του κώδικα.

time_series_MLP.ipynb

```
import pandas as pd
from math import sqrt
import numpy as np
from itertools import chain
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
from numpy import concatenate
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score
```

time_series_RNN.ipynb

```
import pandas as pd
from math import sqrt
import numpy as np
from itertools import chain
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import SimpleRNN, LSTM, GRU
import matplotlib.pyplot as plt
from numpy import concatenate
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error, r2_score
```

Σε επόμενο μπλοκ κώδικα δημιουργούμε την συνάρτηση με όνομα timeseries_to_supervised() και με παραμέτρους df που αποτελούν τα δεδομένα που θέλουμε να μετατρέψουμε, n_in που είναι ο αριθμός των inputs και n_out ο αριθμός των outputs. Τον κώδικα της συνάρτησης τον έχουμε πάρει από τις διαφάνειες του εργαστηρίου. Η ίδια συνάρτηση υπάρχει και στα δύο προαναφερθέντα αρχεία.

```
def timeseries_to_supervised(df, n_in, n_out):
    agg = pd.DataFrame()
    for i in range(n_in, 0, -1):
        df_shifted = df.shift(i).copy()
        df_shifted.rename(columns=lambda x: ('%s(t-%d)' % (x, i)), inplace=True)
        agg = pd.concat([agg, df_shifted], axis=1)
    for i in range(0, n_out):
        df_shifted = df.shift(-i).copy()
        if i == 0:
            df_shifted.rename(columns=lambda x: ('%s(t)' % (x)), inplace=True)
        else:
            df_shifted.rename(columns=lambda x: ('%s(t+%d)' % (x, i)), inplace=True)
        agg = pd.concat([agg, df_shifted], axis=1)
    agg.dropna(inplace=True)
    return agg
```

Σε επόμενο κομμάτι κώδικα, φορτώνουμε τα δεδομένα μας από το Excel αρχείο «data_akbilgic(1).xlsx», το οποίο είναι απαραίτητο να το φορτώσετε στο περιβάλλον του Google Colab, για να τρέξει κανονικά ο κώδικας. Χρησιμοποιούμε ως χαρακτηριστικό στόχο το USD BASED και για αυτό πάιρνουμε τα δεδομένα μας από την στήλη B και μετά. Στην συνέχεια κανονικοποιούμε τα δεδομένα μας σε διάστημα [0,1] και καλούμε την συνάρτηση για την μετατροπή των δεδομένων σε supervised. Στο συγκεκριμένο παράδειγμα απεικονίζονται τα δεδομένα με αριθμό steps ίσο με 6. Σημειώνεται ότι ο ίδιος κώδικας υπάρχει και στα δύο αρχεία.

```
X = pd.read_excel('data_akbilgic(1).xlsx', usecols = "B:I") # loading the data

# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(X)

# convert ndarray to pandas dataframe
scaled = pd.DataFrame(scaled, columns=["data.1", "data.2", "data.3", "data.4", "data.5", "data.6", "data.7", "data.8"])

n_steps = 6
sdf = timeseries_to_supervised(scaled, n_steps, 1)
lst = []
for i in range(1,8):
    lst.append((n_steps+1)*8 - i)
# removing the columns we do not need (SP(t), DAX(t), FTSE(t), NIKKEI(t), BOVESPA(t), EU(t), EM(t))
sdf.drop(sdf.columns[lst], axis=1, inplace=True)
print(sdf)
#sdf.to_excel("test.xlsx")
```

Τα αποτελέσματα του παραπάνω κώδικα φαίνονται στο screenshot:

| | data.1(t-6) | data.2(t-6) | data.3(t-6) | ... | data.7(t-1) | data.8(t-1) | data.1(t) |
|-----|-------------|-------------|-------------|-----|-------------|-------------|-----------|
| 6 | 0.664154 | 0.404333 | 0.489969 | ... | 0.313877 | 0.184496 | 0.455624 |
| 7 | 0.628741 | 0.505990 | 0.546240 | ... | 0.315871 | 0.390618 | 0.265266 |
| 8 | 0.314902 | 0.194024 | 0.310007 | ... | 0.031043 | 0.346048 | 0.363922 |
| 9 | 0.000000 | 0.470147 | 0.364884 | ... | 0.317164 | 0.175245 | 0.631251 |
| 10 | 0.509203 | 0.266894 | 0.291678 | ... | 0.495236 | 0.572906 | 0.217804 |
| .. | ... | ... | ... | ... | ... | ... | ... |
| 525 | 0.409163 | 0.504739 | 0.438167 | ... | 0.492265 | 0.407771 | 0.494245 |
| 526 | 0.647323 | 0.577499 | 0.604836 | ... | 0.455700 | 0.394418 | 0.439530 |
| 527 | 0.533918 | 0.420259 | 0.469508 | ... | 0.395244 | 0.281535 | 0.322631 |
| 528 | 0.306824 | 0.461667 | 0.482772 | ... | 0.389044 | 0.275618 | 0.491722 |
| 529 | 0.532410 | 0.465978 | 0.498359 | ... | 0.462147 | 0.492021 | 0.459275 |

[524 rows x 49 columns]

Τρέχουμε και ένα παράδειγμα με ένα μόνο step για λόγους πληρότητας:

| | data.1(t-1) | data.2(t-1) | data.3(t-1) | ... | data.7(t-1) | data.8(t-1) | data.1(t) |
|-----|-------------|-------------|-------------|-----|-------------|-------------|-----------|
| 1 | 0.664154 | 0.404333 | 0.489969 | ... | 0.530944 | 0.776771 | 0.628741 |
| 2 | 0.628741 | 0.505990 | 0.546240 | ... | 0.519229 | 0.548080 | 0.314902 |
| 3 | 0.314902 | 0.194024 | 0.310007 | ... | 0.273987 | 0.214765 | 0.000000 |
| 4 | 0.000000 | 0.470147 | 0.364884 | ... | 0.373348 | 0.221615 | 0.509203 |
| 5 | 0.509203 | 0.266894 | 0.291678 | ... | 0.326501 | 0.356172 | 0.228529 |
| .. | ... | ... | ... | ... | ... | ... | ... |
| 525 | 0.503240 | 0.493220 | 0.553813 | ... | 0.492265 | 0.407771 | 0.494245 |
| 526 | 0.494245 | 0.476546 | 0.519007 | ... | 0.455700 | 0.394418 | 0.439530 |
| 527 | 0.439530 | 0.419742 | 0.467387 | ... | 0.395244 | 0.281535 | 0.322631 |
| 528 | 0.322631 | 0.448601 | 0.494016 | ... | 0.389044 | 0.275618 | 0.491722 |
| 529 | 0.491722 | 0.487279 | 0.508032 | ... | 0.462147 | 0.492021 | 0.459275 |

[529 rows x 9 columns]

Πλέον, μπορούμε να μετασχηματίσουμε τα δεδομένα σε μορφή [samples, steps, features] και φτιάχνουμε τα train (70%) και test (30%) sets. Αυτό το υλοποιούμε μέσω του παρακάτω κώδικα ο οποίος είναι λίγο διαφορετικός σε κάθε αρχείο.

time series MLP.ipynb

(n steps = 6)

time series RNN.ipynb

```
sdf = sdf.values

# Split data into train and test
len_data = sdf.shape[0]
print("Total length of data:", len_data)
train_size = int(len_data * .7)
print ("Train size: %d" % train_size)
print ("Test size: %d" % (len_data - train_size))

train = sdf[:train_size, :]
test = sdf[train_size:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
sdf = sdf.values

# Split data into train and test
len_data = sdf.shape[0]
print("Total length of data:", len_data)
train_size = int(len_data * .7)
print ("Train size: %d" % train_size)
print ("Test size: %d" % (len_data - train_size))

train = sdf[:train_size, :]
test = sdf[train_size:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

train_y = np.reshape(train_y, (train_size, 1, 1))
test_y = np.reshape(test_y, (len_data - train_size, 1, 1))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
Total length of data: 524  
Train size: 366  
Test size: 158  
(366, 1, 48) (366,) (158, 1, 48) (158,)
```

```
Total length of data: 524  
Train size: 366  
Test size: 158  
(366, 1, 48) (366, 1, 1) (158, 1, 48) (158, 1, 1)
```

Ερώτημα 2: Time-series prediction – υλοποίηση με keras/tensorflow.

Επισημαίνουμε ότι, τα δεδομένα μας έχουν κανονικοποιηθεί και έχουν μετατραπεί στην κατάλληλη μορφή [samples, steps, features] όπως περιγράφθηκαν παραπάνω.

a) Time-series prediction με MLP

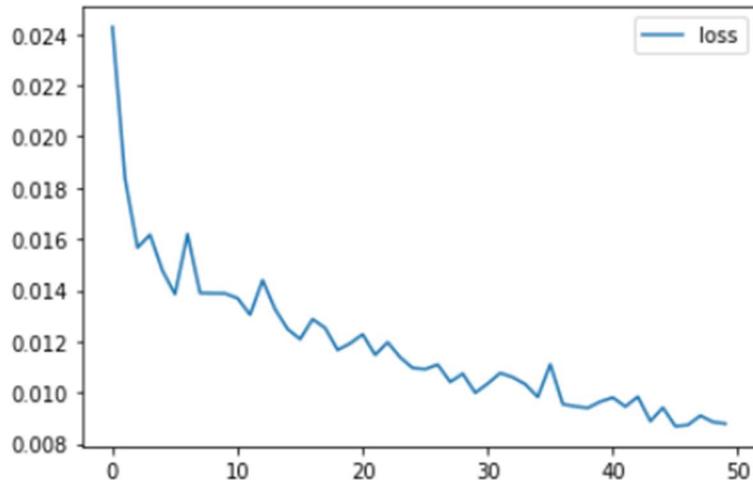
Για την δημιουργία του μοντέλου MLP χρησιμοποιούμε το keras. Ως input layer δίνουμε τις διαστάσεις των steps, features, έχουμε δύο hidden layers το ένα με 300 νευρώνες και το άλλο με 100 και στο output layer έχουμε έναν νευρώνα, αφού προβλέπουμε μια τιμή. Ως loss function χρησιμοποιούμε την mean squared error και optimizer τον adam. Κάνουμε train το μοντέλο και δημιουργούμε ένα γράφημα του loss function.

```
# Building a model with MLP
batch_size = 1 #1
model = Sequential()

model.add(Dense(units=300, input_shape=(train_X.shape[1], train_X.shape[2]), activation="relu"))
model.add(Dense(100, activation="relu"))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

history = model.fit(train_X, train_y, epochs=50, batch_size=batch_size, verbose=2)

# plot history
plt.plot(history.history['loss'], label='loss')
plt.legend()
plt.show()
```



Στη συνέχεια υπολογίζουμε τα εξής metrics:

- Mean squared error: Είναι ο μέσος όρος του τετραγώνου των σφαλμάτων. Το σφάλμα σε αυτή την περίπτωση είναι η διαφορά μεταξύ των παρατηρούμενων τιμών y_1, y_2, y_3, \dots και των προβλεπόμενων τιμών $\text{pred}(y_1), \text{pred}(y_2), \text{pred}(y_3), \dots$. Τετραγωνίζουμε κάθε διαφορά $(\text{pred}(y_n) - y_n)^2$ 2 έτσι ώστε οι αρνητικές και θετικές τιμές να μην αικυρώνουν η μια την άλλη.
- Mean absolute error: Είναι ο μέσος όρος όλων των απόλυτων σφαλμάτων.
- R2 score: Είναι το ποσοστό της διακύμανσης στην εξαρτημένη μεταβλητή που μπορεί να προβλεφθεί από τις ανεξάρτητες μεταβλητές.

Και τέλος, εκτυπώνουμε ένα γράφημα που διακρίνονται τα original δεδομένα (με πράσινο χρώμα) σε σχέση με τα προβλέψιμα δεδομένα (με κόκκινο χρώμα) και μια μπλε γραμμή που χωρίζει το γράφημα στο training set και στο testing set.

```

# make a prediction
testPredict = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))

# invert scaling for prediction
testPredict = np.reshape(testPredict, (testPredict.shape[0], 1))
inv_testPredict = concatenate((testPredict, test_X[:, -7:]), axis=1)
inv_testPredict = scaler.inverse_transform(inv_testPredict)
inv_testPredict = inv_testPredict[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
test_y = concatenate((test_y, test_X[:, -7:]), axis=1)
test_y = scaler.inverse_transform(test_y)
test_y = test_y[:,0]

# calculate error
print("Test MSE: ", mean_squared_error(test_y, inv_testPredict))
print("Test MAE: ", sum(abs(test_y-inv_testPredict))/test_y.shape[0])
print("Test R2: ", r2_score(test_y, inv_testPredict))

```

```

# Finally, we check the result in a plot.
# A vertical line in a plot identifies a splitting point between
# the training and the test part.
predicted = np.concatenate((inv_trainPredict,inv_testPredict),axis=0)

original = np.concatenate((train_y,test_y),axis=0)
predicted = np.concatenate((inv_trainPredict,inv_testPredict),axis=0)
index = range(0, original.shape[0])
plt.plot(index,original, 'g')
plt.plot(index,predicted, 'r')
plt.axvline(scaled.index[train_size], c="b")
plt.show()

```

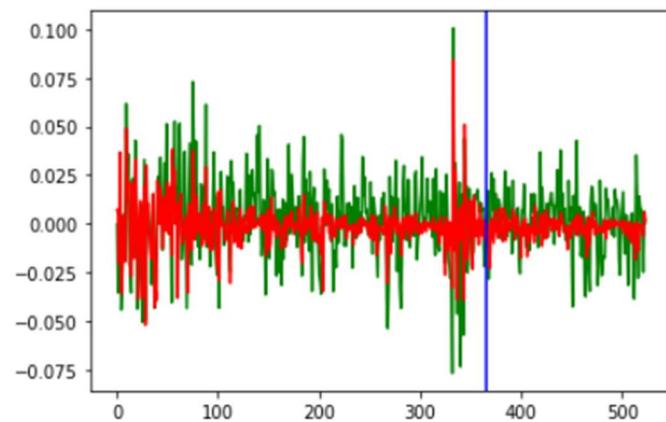
```

# invert scaling for prediction
trainPredict = np.reshape(trainPredict, (trainPredict.shape[0], 1))
inv_trainPredict = concatenate((trainPredict, train_X[:, -7:]), axis=1)
inv_trainPredict = scaler.inverse_transform(inv_trainPredict)
inv_trainPredict = inv_trainPredict[:,0]

# invert scaling for actual
train_y = train_y.reshape((len(train_y), 1))
train_y = concatenate((train_y, train_X[:, -7:]), axis=1)
train_y = scaler.inverse_transform(train_y)
train_y = train_y[:,0]

Test MSE:  0.00023483997430499024
Test MAE:  0.012037471360784275
Test R2:  0.008184730009769492

```



Βλέπουμε από το τελευταίο γράφημα, ότι δεν είναι τόσο καλά τα αποτελέσματα. Όπως μας είπατε και στο εργαστήριο, το dataset που έχουμε είναι πολύ δύσκολο να προβλεφθεί και γενικά τέτοιου είδους προβλήματα είναι πολύ δύσκολα, για αυτό δεν έχουμε τόσο καλά αποτελέσματα.

Αλλάζοντας των αριθμό των steps ή το πλήθος των νευρώνων στα hidden layers, δεν είχε κάποια σημαντική διαφορά στα αποτελέσματα.

Η χρήση ενός MLP σε ένα time series πρόβλημα δεν είναι η κατάλληλη, γιατί δεν θα μπορούσαμε να μάθουμε την εξάρτηση ενός επιπέδου από το προηγούμενο επίπεδο. Η έξοδος θα μπορούσε απλώς να προβλεφθεί ως τον συνδυασμό εισόδων μετά το training και ως εκ τούτου, δεν θα είμαστε σε θέση να προσδιορίσουμε πώς κάθε βήμα οδηγεί σε αλλαγή. Η χρήση ενός MLP με αυτόν τον τρόπο ουσιαστικά σημαίνει ότι, δεν εξετάζουμε την αλλαγή σε μια ακολουθία σε σχέση με το χρόνο.

b) Time-series prediction με RNN

Για την δημιουργία του μοντέλου RNN χρησιμοποιούμε το keras. Ως input layer δίνουμε τις διαστάσεις των steps, features, έχουμε δύο hidden layers το ένα με 300 νευρώνες και το άλλο με 100 και στο output layer έχουμε έναν νευρώνα, αφού προβλέπουμε μια τιμή. Ως loss function χρησιμοποιούμε την mean squared error και optimizer το adam. Κάνουμε train το μοντέλο και δημιουργούμε ένα γράφημα του loss function.

```

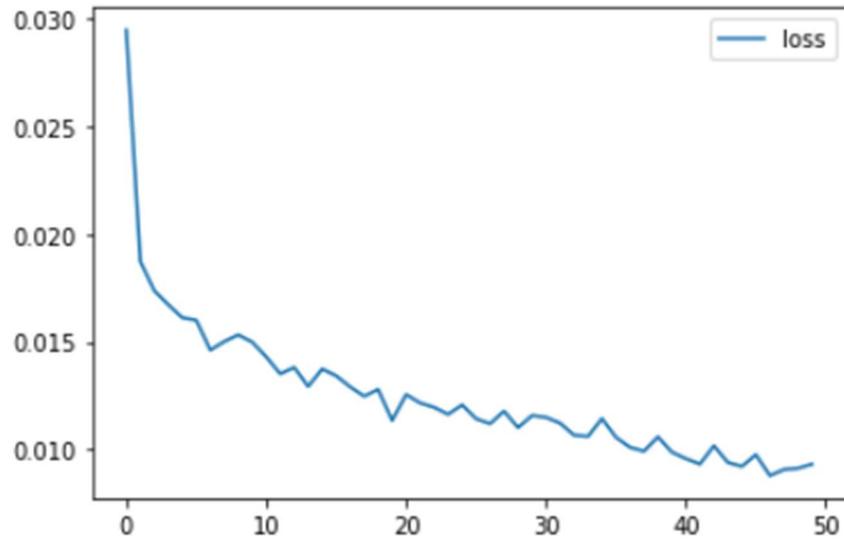
# Building a model with SimpleRNN
batch_size = 1 #1
model = Sequential()

model.add(SimpleRNN(units=300, input_shape=(train_X.shape[1], train_X.shape[2]), activation="relu", return_sequences=True))
model.add(Dense(100, activation="relu"))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

history = model.fit(train_X, train_y, epochs=50, batch_size=batch_size, verbose=2)

# plot history
plt.plot(history.history['loss'], label='loss')
plt.legend()
plt.show()

```



Στη συνέχεια υπολογίζουμε τα ίδια metrics με αυτά που υπολογίσαμε και στο MLP και τέλος, εκτυπώνουμε ένα γράφημα που διακρίνονται τα original δεδομένα (με πράσινο χρώμα) σε σχέση με τα προβλέψιμα δεδομένα (με κόκκινο χρώμα) και μια μπλε γραμμή που χωρίζει το γράφημα στο training set και το testing set.

```

# make a prediction
testPredict = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))

# invert scaling for prediction
testPredict = np.reshape(testPredict, (testPredict.shape[0], 1))
inv_testPredict = concatenate((testPredict, test_X[:, -7:]), axis=1)
inv_testPredict = scaler.inverse_transform(inv_testPredict)
inv_testPredict = inv_testPredict[:,0]

# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
test_y = concatenate((test_y, test_X[:, -7:]), axis=1)
test_y = scaler.inverse_transform(test_y)
test_y = test_y[:,0]

# calculate error
print("Test MSE: ", mean_squared_error(test_y, inv_testPredict))
print("Test MAE: ", sum(abs(test_y-inv_testPredict))/test_y.shape[0])
print("Test R2: ", r2_score(test_y, inv_testPredict))

# Finally, we check the result in a plot.
# A vertical line in a plot identifies a splitting point between
# the training and the test part.
predicted = np.concatenate((inv_trainPredict,inv_testPredict),axis=0)

original = np.concatenate((train_y,test_y),axis=0)
predicted = np.concatenate((inv_trainPredict,inv_testPredict),axis=0)
index = range(0, original.shape[0])
plt.plot(index,original, 'g')
plt.plot(index,predicted, 'r')
plt.axvline(scaled.index[train_size], c="b")
plt.show()

```

```

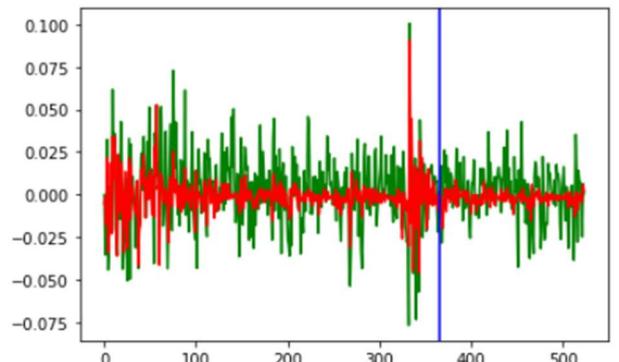
trainPredict = model.predict(train_X)
train_X = train_X.reshape((train_X.shape[0], train_X.shape[2]))

# invert scaling for prediction
trainPredict = np.reshape(trainPredict, (trainPredict.shape[0], 1))
inv_trainPredict = concatenate((trainPredict, train_X[:, -7:]), axis=1)
inv_trainPredict = scaler.inverse_transform(inv_trainPredict)
inv_trainPredict = inv_trainPredict[:,0]

# invert scaling for actual
train_y = train_y.reshape((len(train_y), 1))
train_y = concatenate((train_y, train_X[:, -7:]), axis=1)
train_y = scaler.inverse_transform(train_y)
train_y = train_y[:,0]

Test MSE:  0.0002313789868860198
Test MAE:  0.011981514238679063
Test R2:  0.02280174817943137

```



Βλέπουμε ξανά από το τελευταίο γράφημα, ότι δεν είναι τόσο καλά τα αποτελέσματα. Ωστόσο, αλλάζοντας των αριθμών steps είχε μια αρκετά μικρή επίδραση... Αν κοιτάξουμε τα metrics ανάμεσα στο RNN και στο MLP για αριθμό step = 6, θα δούμε ότι στα MSE και MEA το RNN έχει σχετικά μικρότερο σφάλμα από το MLP, ενώ στο R2 score ισχύει το ανάποδο.

Τα RNN μαθαίνουν πώς μια ακολουθία αλλάζει με το χρόνο. Επίσης, λόγω της επαναλαμβανόμενης διαδικασίας στα RNN, η έξοδος ενός συγκεκριμένου σταδίου μπορεί να τροφοδοτηθεί ξανά ως είσοδος και έτσι, ένα RNN μπορεί να μάθει πώς μια μικρή αλλαγή σε ένα πολύ προηγούμενο στάδιο προκάλεσε την έξοδο να αλλάξει, κατά συνέπεια, ένας συντελεστής χρόνου υπάρχει στα RNN.

Συμπεραίνοντας, καταλαβαίνουμε ότι τα RNN είναι πιο κατάλληλα μοντέλα για να χρησιμοποιηθούν σε ένα πρόβλημα χρονοσειράς.

ΠΗΓΕΣ:

Όλο το υλικό εργαστηρίου (διαφάνειες, κώδικες)

https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

<https://www.oreilly.com/library/view/mastering-machine-learning/9781788621113/f4a7b48b-0021-4917-ad0d-e89cef3b4df6.xhtml>

<https://www.oreilly.com/library/view/mastering-machine-learning/9781788621113/2830f738-c6a5-460a-b518-23ecd3745c2d.xhtml>

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html#sklearn.metrics.completeness_score

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html#sklearn.metrics.homogeneity_score

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cluster.pair_confusion_matrix.html

<https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/>

<https://www.bmc.com/blogs/mean-squared-error-r2-and-variance-in-regression-analysis/>

<https://www.statisticshowto.com/absolute-error/>

<https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/>

<https://www.quora.com/In-using-deep-learning-for-time-series-analysis-what-is-the-difference-in-using-RNN-vs-vanilla-MLPs>