Software Development for Algorithmic Problems Project-3

(Ομάδα Χρηστών 66)

Μέλη Ομάδας

Ονοματεπώνυμο	Αριθμός Μητρώου
Ιωάννης Καπετανγεώργης	1115201800061
Δημήτριος Σιταράς	1115201800178

Github link:
https://github.com/giannhskp/Software-Development-for-Algorithmic-Problems_Project-3

Οργάνωση Κώδικα

Γενικά

- Ο κώδικας είναι σχολιασμένος.
- Πληρούνται όλες οι προϋποθέσεις / απαιτήσεις που αναγράφονται στην εκφώνηση της άσκησης.
- Η υλοποίηση του project έγινε με τη χρήση συστήματος διαχείρισης εκδόσεων λογισμικού και συνεργασίας (Git).
- Για την εκτέλεση των python αρχείων πρέπει να προηγηθεί η εγκατάσταση των βιβλιοθηκών που χρησιμοποιούνται, έτσι έχουμε προσθέσει το αρχείο packages.txt που περιέχει τις αντίστοιχες βιβλιοθήκες που χρειάζονται. Συνεπώς, αρκεί να τρέξουμε στο τερματικό την εντολή *pip install -r packages.txt* προκειμένου να γίνει η εγκατάστασή τους (για την απεικόνιση των γραφικών παραστάσεων ίσως χρειαστεί και η εγκατάσταση της ipython, ώστε να γίνεται με αυτή η εκτέλεση των αρχείων).
- Εντολές εκτέλεσης (ακριβώς όπως αναγράφονται στην εκφώνηση):
    - *python/ipython forecast.py -d <dataset> -n <number of time series selected>*
        - *π.χ. python forecast.py -d input_files/nasdaq2007_17.csv -n 15*
        - *ipython forecast.py -- -d input_files\nasdaq2007_17.csv -n 15*
    - *python/ipython detect.py -d <dataset> -n <number of time series selected> -mae <error value as double>*
        - *π.χ. python detect.py -d input_files/nasdaq2007_17.csv -n 15*
        - *ipython detect.py -- -d input_files\nasdaq2007_17.csv -n 15*
    - *python/ipython reduce.py -d <dataset> -q <queryset> -od <output_dataset_file> -oq <output_query_file>*
        - *π.χ. python reduce.py -d input_files/part3_dataset.csv -q input_files/part3_query.csv -od output_dataset.csv -oq output_query.csv*
        - *ipython reduce.py -- -d input_files\part3_dataset.csv -q input_files\part3_query.csv -od output_dataset.csv -oq output_query.csv*

- Στον φάκελο με όνομα “models” έχουμε προσθέσει μοντέλα για το κάθε ερώτημα, τα οποία έχουν εκπαιδευτεί με τα αντίστοιχα train sets στο Google Colab. Συγκεκριμένα:

    - Για το Α ερώτημα (models/part1) έχουμε συμπεριλάβει ένα μοντέλο που έχει εκπαιδευτεί με ολόκληρο το σύνολο των curves και 20 μοντέλα (models/part1/one_curve_models) που κάθε ένα έχει εκπαιδευτεί με ένα μόνο curve, έχουν χρησιμοποιηθεί τα 20 πρώτα από το dataset .Η υλοποίηση των μοντέλων είναι η ίδια απλά διαφέρουν ως προς το σύνολο εκπαίδευσης και τις υπερ-παραμέτρους.
    - Για το Β ερώτημα (models/part2) έχουμε συμπεριλάβει ένα μοντέλο (autoencoder) που έχει εκπαιδευτεί με τα 200 πρώτα curves από το dataset.
    - Για το Γ ερώτημα (models/part3) έχουμε συμπεριλάβει δύο μοντέλα (autoencoder και encoder) που έχουν εκπαιδευτεί με όλες τις curves από το dataset.

Μέρος Α - Time Series Forecasting

Σε αυτό το ερώτημα υλοποιούμε ένα αναδρομικό νευρωνικό δίκτυο LSTM πολλαπλών στρωμάτων το οποίο χρησιμοποιείται για την πρόγνωση τιμών των αντίστοιχων μετοχών που δίνονται ως είσοδο.
Αναλυτικότερα, έχουμε υλοποιήσει, 2 μοντέλα, το ένα εκπαιδεύεται με όλες τις χρονοσειρές από το dataset και το άλλο εκπαιδεύεται με μία μόνο. Η δομή του μοντέλου που χρησιμοποιήθηκε για την εκπαίδευση ανα σύνολο χρονοσειρών και ανα χρονοσειρά είναι η ίδια και στις δύο περιπτώσεις. Ωστόσο υπάρχει διαφορά στις τιμές των υπερ-παραμέτρων που επιλέχθηκαν για την εκπαίδευση.

Διαχωρισμός των δεδομένων σε train και test set

Σε αρχικό στάδιο (και στα 2 μοντέλα) χωρίζουμε τις μετοχές σε training και σε test set ανάλογα με το μήκος των χρονοσειρών, στο training set έχουμε συμπεριλάβει τις πρώτες 1825 τιμές από τις δεδομένες χρονοσειρές και στο test set έχουμε τις υπόλοιπες 1825 τιμές για τις ίδιες βεβαια χρονοσειρές (με λίγα λόγια έχουν χωριστεί στην μέση, 50-50).
Έχοντας ορίσει τα train και test set, χρησιμοποιούμε το train set για την εκπαίδευση του μοντέλου και το test set για να δοκιμάσουμε την απόδοση του μοντέλου μας έπειτα από το πέρας της εκπαιδευσης, δίνοντας του “άγνωστα” δεδομένα.

Ορισμός υπερ-παραμέτρων

Όπως αναφέρεται και στην συνέχεια, για την εκπαιδευση του μοντέλου χρησιμοποιούμε ένα μικρό υποσύνολο του train set σαν validation set.
Μέσω του validation set μπορούμε να παρακολουθήσουμε την συμπεριφορά του μοντέλου μας σε ¨άγνωστα¨ δεδομένα κατά την διάρκεια εκπαίδευσης του και να το τροποποιήσουμε έτσι ώστε να αποδίδει εξίσου καλά και στο validation set.
Πιο συγκεκριμένα, παρακολουθώντας το training loss και το validation loss για κάθε ένα epoch, και κατασκευάζοντας τα αντίστοιχα loss vs epochs curves μπορούμε να παρατηρήσουμε την συμπεριφορά του μοντέλου κατά την εκπαίδευση και να τροποποιήσουμε τις υπερ-παραμέτρους του μοντέλου έτσι ώστε να αυξήσουμε την απόδοση του ενώ ταυτόχρονα να εξαλείψουμε την παρουσία overfit ή underfit στο μοντέλο μας.

Έτσι, έπειτα από τους αντίστοιχους πειραματισμούς, ορίζουμε τα παρακάτω hyperparameters:

- Για το μοντέλο το οποίο εκπαιδεύεται από όλες τις μετοχές που υπάρχουν στο dataset:
    - EPOCHES = 5, λόγω του μεγάλου πλήθους των μετοχών του train set το μοντέλο δεν χρειάζεται πολλά epochs για να εκπαιδευτεί. Δοκιμάζοντας μεγαλύτερο αριθμό epochs η εκπαίδευση του μοντέλου καθυστερούσε υπερβολικά πολύ ενώ ταυτόχρονα τόσο το train όσο και το validation loss παρέμεναν σταθερά (ή είχαν αμελητέα μείωση). Στον βέλτιστο αριθμό epochs καταλήξαμε χρησιμοποιώντας την τεχνική Early stopping κάνοντας monitor το validation loss και σταματώντας την εκπαίδευση όταν αυτό αρχίσει να αυξάνεται. Έτσι αποφευγουμε το overfit στο μοντέλο μας.
    - BATCH_SIZE = 2048, γνωρίζουμε πως όσο μεγαλύτερο είναι το batch size τόσο πιο λίγο θόρυβο προκαλεί και τόσο καλύτερη γίνεται η εκτίμηση. Δοκιμάζοντας πιο μικρά μεγέθη (συνήθως δυνάμεις του 2) παρατηρήσαμε πως η εκπαίδευση καθυστερούσε αρκετά και τα αποτελέσματα δεν ήταν ικανοποιητικά, ενώ για μεγαλύτερα δεν παρατηρήθηκαν διαφορές.
    - LEARNING_RATE = 0.01, δοκιμάσαμε μεγαλύτερες τιμές, στις οποίες το μοντέλο λόγω της πιο γρήγορης εκμάθησης κατα την διάρκεια της εκπαίδευσης παρουσίαζε αστάθεια στα αποτελέσματα και πολλές φορές το μοντέλο παρουσίαζε overfit. Τα παραπάνω επιβεβαιώνονται παρακολουθώντας την πορεία των train και validation loss curves. Δοκιμάζοντας μικρότερες τιμές, το μοντέλο χρειαζόταν περισσότερα epochs για να εκπαιδευτεί και τελικά δεν είχε την επιθυμητή απόδοση..
    - WINDOW SIZE = 60, η οποία παράμετρος αφορά ουσιαστικά τα time steps που θα χωριστούν οι μετοχές του train set, ώστε στη συνέχεια να φτιάξουμε τα κατάλληλα σύνολα (X_train και y_train) που θα χρησιμοποιηθούν για την εκπαίδευση του μοντέλου. Δηλαδή το μοντέλο μέσω 60 συνεχόμενων τιμών μιας χρονοσειράς θα προβλέπει την αμέσως επόμενη τιμή. Δοκιμάσαμε τιμές από 10 έως 70, χωρίς να παρατηρούμε μεγάλες διαφορές, τελικά καταλήξαμε να βάλουμε την τιμή 60 που μας προέκυπταν τα καλύτερα αποτελέσματα.

- Για το μοντέλο που εκπαιδεύεται με μόνο μια μετοχή από το dataset:
    - EPOCHES = 20, αρκούν για να επιτευχθεί σύγκλιση και σταθεροποίηση των train και validation loss.
    - BATCH_SIZE = 16, λόγω του πολύ μικρού όγκου του train set (μια μόνο χρονοσειρά) επιλέξαμε μια αντίστοιχα μικρή τιμή για το batch size με την οποία παρατηρήσαμε πολύ καλή συμπεριφορά του μοντέλου.

- LEARNING_RATE = 0.001, δοκιμάσαμε μεγαλύτερες τιμές, στις οποίες το μοντέλο λόγω της πιο γρήγορης εκμάθησης κατα την διάρκεια της εκπαίδευσης και του μικρού όγκου των δεδομένων παρουσίαζε αρκετή αστάθεια και αρκετό overfit, αλλά και μικρότερες τιμές, στις οποίες η εκπαίδευση καθυστερούσε υπερβολικά χωρίς τελικά το μοντέλο να έχει την επιθυμητή απόδοση.
- WINDOW SIZE = 60, η οποία παράμετρος αφορά ουσιαστικά τα time steps που θα χωριστούν οι μετοχές του train set, ώστε στη συνέχεια να φτιάξουμε τα κατάλληλα σύνολα (X_train και y_train) που θα χρησιμοποιηθούν για την εκπαίδευση του μοντέλου. Δοκιμάσαμε τιμές από 10 έως 70, χωρίς να παρατηρούμε μεγάλες διαφορές, τελικά καταλήξαμε να βάλουμε την τιμή 60 που μας προέκυπταν τα καλύτερα αποτελέσματα.

Προ-επεξεργασία δεδομένων - Scaling

Στη συνέχεια, κατασκευάζουμε την δομή που θα χρησιμοποιηθεί για την εκπαίδευση του μοντέλου (δηλαδή τα σύνολα X_train και y_train) εφαρμόζοντας ταυτόχρονα και το απαραίτητο scaling (ώστε να κανονικοποιήσουμε τις τιμές των μετοχών).
Όπως έχει αναφερθεί και παραπάνω το train set (και κατ' επέκταση τα X_train και y_train) κατασκευάζεται από το πρώτο μισό της κάθε χρονοσειράς. Έτσι στην παράγραφο αυτή, όταν αναφερόμαστε σε μία χρονοσειρά εννοούμε στο πρώτο μισό της χρονοσειράς.
Για την παραγωγή του X_train οι χρονοσειρές χωρίζονται σε διαδοχικά sets μήκους ίσου με το WINDOW SIZE. Πιο συγκεκριμένα το πρώτο set τιμών αποτελείται από τις πρώτες window τιμές της χρονοσειράς, στο δεύτερο set έχει αφαιρεθεί η πρώτη τιμή του πρώτου set και έχει προστεθεί η window+1 τιμή της χρονοσειράς κ.ο.κ.. Το τελευταίο set περιέχει window τιμές με την τελευταία τιμή του set να αντιστοιχεί στην προ-τελευταία τιμή της χρονοσειράς. Δηλαδή τα windows είναι επικαλυπτώμενα.
Σε κάθε ένα set window τιμών του X_train αντιστοιχεί μια τιμή στο y_train. Η τιμή αυτή είναι η αμέσως επόμενη τιμή της χρονοσειράς την οποία θέλουμε το μοντέλο μας να προβλέψει.
Στην περίπτωση της εκπαίδευσης ανα σύνολο χρονοσειρών το X_train αποτελείται από sets όλων των χρονοσειρών και το y_train αντίστοιχα. Όπως είναι προφανές, κανένα set δεν αποτελείται από τιμές δύο διαφορετικών χρονοσειρών.
Για το scale χρησιμοποιούμε τον MinMaxScaler, επιπλέον δοκιμάσαμε και τον StandardScaler καθώς γενικότερα γνωρίζουμε πως οι μετοχές ακολουθούν την κανονική κατανομή, όμως οι προβλέψεις του μοντέλου μας δεν ήταν τόσο ικανοποιητικές.
Όσον αφορά το scaling, εφαρμόζουμε ξεχωριστό scaling σε κάθε ένα window (συγκεκριμένα ανά 60 time steps). Δηλαδή ο scaler γίνεται fit σε κάθε ένα window

ξεχωριστά. Το εύρος τιμών εσωτερικά ενός window είναι σχετικά μικρό. Έτσι οι scaled τιμές που προκύπτουν για κάθε ένα window είναι ομοιόμορφες σε κάθε περίπτωση. Πρόσθετα, πειραματιστήκαμε κάνοντας και scale ανά μετοχή, συγκρίνοντας τα αποτελέσματα του μοντέλου με το scale ανα window, στο οποίο τελικά καταλήξαμε αφού οι αντίστοιχες προβλέψεις ήταν σαφώς καλύτερες. Το scale ανά μετοχή δεν είναι τόσο αποδοτικό, διότι η αναπαράσταση που προκύπτει δεν αντιπροσωπεύει ακριβώς τις αρχικές τιμές της κάθε μετοχής, καθώς ορισμένες μετοχές έχουν μεγάλο εύρος τιμών ενώ άλλες έχουν πολύ μικρό με αποτέλεσμα οι scaled τιμές που προκύπτουν να μην είναι ομοιόμορφες μεταξύ των διαφορετικών χρονοσειρών.

Ορισμός του αναδρομικού νευρωνικού δικτύου

Έπειτα, το νευρωνικό που υλοποιήσαμε αποτελείται 4 LSTM layers με units που μειώνονται σταδιακά από layer σε layer, δηλαδή 128 -> 96 -> 64 ->32, καθώς γνωρίζουμε πως όσο πιο ¨βαθύ¨ είναι ένα μοντέλο τόσο μεγαλύτερη πιθανότητα έχει να αποδώσει καλύτερα, έχοντας ένα μεγάλο όγκο δεδομένων για εκπαίδευση.
Βέβαια, δοκιμάσαμε μοντέλα με πιο λίγα layers, με σταθερό αριθμό από units καθώς επίσης πειραματιστήκαμε και με διάφορα activation functions, όμως καμία από αυτές τις υλοποιήσεις δεν μας έδινε τόσο ακριβής προβλέψεις.
Έπειτα από τα LSTM layers, ακολουθεί ένα γραμμικό output layer μέσω του οποίου προκύπτει η πρόβλεψη του μοντέλου. Το γραμμικό αυτό layer δέχεται είσοδο μεγέθους 32 (όσο δηλαδή και το output του τελευταίου LSTM layer) και παράγει 1 output.
Τέλος, προσθέσαμε και ενα Dropout Layer έπειτα από κάθε LSTM layer έτσι ώστε να αποτρέψουμε το μοντέλο από το να παρουσιάσει overfitting. Το dropout rate σε κάθε περίπτωση έχει οριστεί σε 0.2.
Αναλυτικότερα η εικόνα του μοντέλου, όπως παρουσιάζεται από την συνάρτηση model.summary(), είναι η εξής:

```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 60, 128)           66560

dropout_4 (Dropout)          (None, 60, 128)           0

lstm_5 (LSTM)                (None, 60, 96)            86400

dropout_5 (Dropout)          (None, 60, 96)            0

lstm_6 (LSTM)                (None, 60, 64)            41216

dropout_6 (Dropout)          (None, 60, 64)            0

lstm_7 (LSTM)                (None, 32)                12416

dropout_7 (Dropout)          (None, 32)                0

dense_1 (Dense)              (None, 1)                 33

=================================================================
Total params: 206,625
Trainable params: 206,625
Non-trainable params: 0
_________________________________________________________________
```

Έτσι, το μοντέλο μας δέχεται σαν input WINDOW διαδοχικές τιμές μιας χρονοσειράς και παράγει σαν output μια τιμή-πρόβλεψη.

Προφανώς η εκπαίδευση γίνεται μέσω batches, δηλαδή κάθε φορά δέχεται σαν input batch_size sets μεγέθους window_size και παράγει σαν output batch_size τιμές-προβλέψεις.

Εκπαίδευση του μοντέλου

Όσον αφορά την εκπαιδευση του μοντέλου αρχικά ορίζουμε τον optimizer καθώς και το loss function που θα χρησιμοποιηθούν. Πιο συγκεκριμένα, ορίζουμε:

- Optimizer : Adam , δίνοντας του σαν παράμετρο το learning rate το οποίο ορίσαμε παραπάνω.
- Loss Function : Mean Squared Error

Επίσης κατά την κλήση της συνάρτησης fit (η οποία πραγματοποιεί την εκπαίδευση), ορίζουμε τις εξής παραμέτρους:

- X_train και y_train. Το X_train αντιστοιχεί στα sets window τιμών μέσω των οποίων θα προκύπτει η πρόβλεψη του μοντέλου μας. Το y_train αντιστοιχεί στην σωστή πρόβλεψη που πρέπει να κάνει το μοντέλο μας και μέσω της οποίας θα υπολογίζεται κάθε φορά το loss της πρόβλεψης του μοντέλου.
- epochs και batch_size τα οποία έχουν οριστεί στον Ορισμός υπερ-παραμέτρων.
- validation_split=0.1 , δηλαδή ορίζουμε το 10% του train set να θεωρηθεί σαν validation set έτσι ώστε να παρακολουθούμε την πορεία του μοντέλου κατα την εκπαιδευση του πάνω σε άγνωστα δεδομένα. Έτσι, καταφέρνουμε να γενικεύσουμε το μοντέλο μας, δηλαδή να μην παρουσιάζει overfit ή underfit.
- Early stopping με βάση το validation loss και patience = 3. Πιο συγκεκριμένα παρακολουθείται το validation loss και σε περίπτωση που το validation loss αυξάνεται σε τρία διαδοχικά epochs τότε η εκπαίδευση σταματά καθώς το μοντέλο δεν βελτιώνει πλέον την απόδοση του σε άγνωστα δεδομένα, δηλαδή αρχίζει να παρουσιάζει overfit.

Εκπαιδευμένα μοντέλα

Με βάση τα παραπάνω έχουμε εκπαιδεύσει τα εξής μοντέλα:

- Ένα μοντέλο το οποίο έχει εκπαιδευτεί χρησιμοποιώντας όλες τις χρονοσειρές του dataset σαν σύνολο εκπαίδευσης.
- 20 μοντέλα τα οποία έχουν εκπαιδευτεί με μία μόνο χρονοσειρά. Τα 20 αυτά μοντέλα έχουν εκπαιδευτεί με τις 20 πρώτες χρονοσειρές αντίστοιχα.

Τα μοντέλα αυτά τα έχουμε αποθηκεύσει στα αντίστοιχα αρχεία και χρησιμοποιούνται από το εκτελέσιμο αρχείο του πρώτου μέρους της εργασίας (όπως εξηγείται και στην συνέχεια).

Αξιολόγηση του μοντέλου

Όπως αναφέρθηκε και παραπάνω η εκπαίδευση του μοντέλου γίνεται χρησιμοποιώντας το πρώτο μισό του κάθε curve, δηλαδή τις πρώτες 1825 τιμές. Στην περίπτωση της εκπαίδευσης ανα σύνολο των χρονοσειρών χρησιμοποιούνται οι 1825 πρώτες τιμές από όλα τα curves του dataset, ενώ στην περίπτωση της εκπαίδευσης ανα χρονοσειρά χρησιμοποιούνται οι 1825 πρώτες τιμές ενός συγκεκριμένου curve.
Έχοντας το εκπαιδευμένο μοντέλο και ανάλογα με την παράμετρο -n την οποία έχει ορίσει ο χρήστης κατά την εκτέλεση του προγράμματος εκτελείται η ακόλουθη διαδικασία για την αξιολόγηση του μοντέλου.
Για κάθε ένα μια τις n πρώτες χρονοσειρές του dataset:

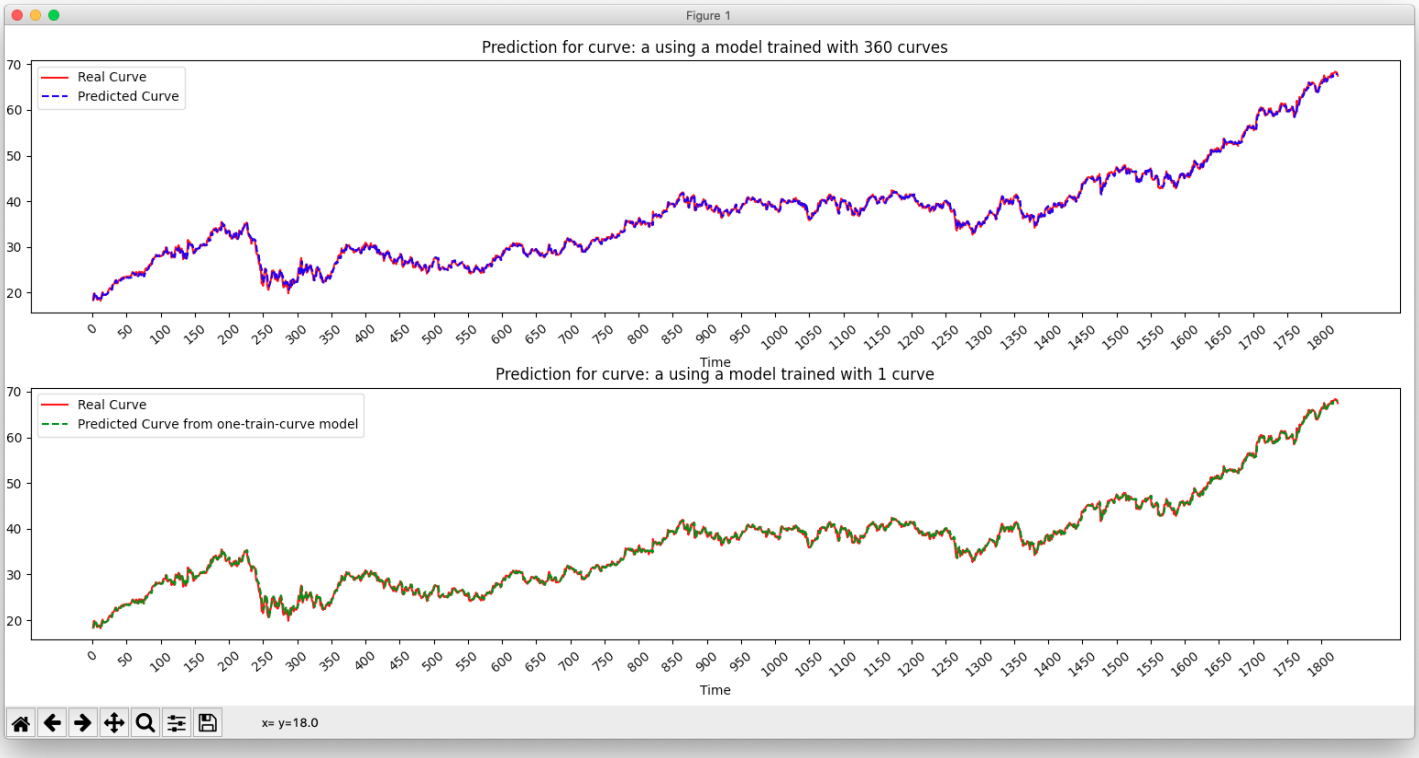
- Χρησιμοποιώντας το δεύτερο μισό της χρονοσειράς, δηλαδή τις 1825 τελευταίες τιμές της δημιουργούμε τα σύνολο X_test . Το σύνολο αυτό δημιουργείται ακριβώς με τον ίδιο τρόπο που δημιουργείται το σύνολο X_train και εξηγείται αναλυτικά στην παράγραφο Προ-επεξεργασία δεδομένων - Scaling. Δηλαδή δημιουργούμε sets μήκους 60 (window_size) από διαδοχικές τιμές της χρονοσειράς. Όπως και στο train set, εφαρμόζουμε scaling ανά window.
- Έτσι το σύνολο X_test αντιστοιχεί στο input που θα δοθεί στο μοντέλο μας έτσι ώστε να δοκιμάσουμε την απόδοση του. Το X_test αποτελείται από 60αδες διαδοχικών τιμών στις οποίες έχει εφαρμοστεί (ξεχωριστά σε κάθε μια) το αντίστοιχο scaling. Για κάθε μια 60αδα το μοντέλο μας παράγει σαν output μια τιμή η οποία αντιστοιχεί στην πρόβλεψη για την αμέσως επόμενη τιμή (δηλαδή την 61η).
- Εκτελώντας την συνάρτηση predict στο μοντέλο μας και δίνοντας σαν είσοδο το X_test, λαμβάνουμε σαν output τόσες τιμές όσος και ο αριθμός των windows που περιέχονται στο X_test. Κάθε μία τιμή αντιστοιχεί στο prediction του αντίστοιχου window με την σειρά που εμφανίζονται.
- Στην συνέχεια κάνουμε unscale (μέσω της inverse_transform) τα output του μοντέλου. Για το unsacling μιας πρόβλεψης χρησιμοποιούμε κάθε φορά τον αντίστοιχο scaler που χρησιμοποιήθηκε στο window που αντιστοιχεί αυτή η πρόβλεψη. Για να γίνει αυτό, κατα την δημιουργία του X_test αποθηκεύουμε για κάθε ένα window τον scaler ο οποίος χρησιμοποιήθηκε και έγινε fit με το συγκεκριμένο window. Έχοντας κάνει unscale όλες τις τιμές-output που μας επέστρεψε το μοντέλο έχουμε κατασκευάσει το δεύτερο μισό της χρονοσειράς μέσω των προβλέψεων του μοντέλου.
- Τέλος παρουσιάζουμε γραφικά την αρχική καμπύλη του dataset και την καμπύλη που προέκυψε μέσω των προβλέψεων του μοντέλου στο ίδιο διάγραμμα. Έτσι μπορούμε να ελέγξουμε την ακρίβεια τον προβλέψεων συγκρίνοντας τις δύο καμπύλες καμπύλες μεταξύ τους (αρχική καμπύλη και καμπύλη προβλέψεων).

Για κάθε μια χρονοσειρά, η παραπάνω διαδικασία εκτελείται τόσο χρησιμοποιώντας το μοντέλο το οποίο έχει εκπαιδευτεί με ολόκληρο το σύνολο των χρονοσειρών όσο και με το μοντέλο το οποίο έχει εκπαιδευτεί με μία μόνο χρονοσειρά.
Έτσι, για μία χρονοσειρά προκύπτουν δύο γραφήματα (ένα για κάθε μοντέλο).

Όπως ήδη αναφέρθηκε, έχουμε εκπαιδεύσει 20 διαφορετικά μοντέλα τα οποία εκπαιδεύτηκαν μέσω μίας μόνο χρονοσειράς. Λέγονας διαφορετικά μοντέλα, εννοούμε πως έχει χρησιμοποιηθεί διαφορετική χρονοσειρά για την εκπαίδευση σε κάθε ένα από αυτά. Συγκεκριμένα έχουν χρησιμοποιηθεί οι 20 πρώτες χρονοσειρές του dataset.
Λόγω του περιορισμένου πλήθους των μοντέλων ανά χρονοσειρά:

- Αν η παράμετρος -n που δόθηκε από τον χρήστη είναι μικρότερη από το 20, τότε και για τις n χρονοσειρές χρησιμοποιείται τόσο το μοντέλο εκπαιδευμένο ανά χρονοσειρά όσο και το μοντέλο εκπαιδευμένο ανα σύνολο χρονοσειρών για την πρόβλεψη του δεύτερου μισού της χρονοσειράς. Δηλαδή ακολουθείται η διαδικασία που εξηγήθηκε παραπάνω και παρουσιάζεται στην συνέχεια. Για κάθε χρονοσειρά χρησιμοποιείται το αντίστοιχο μοντέλο εκπαιδευμένο ανα χρονοσειρά (π.χ. για την n=1 χρονοσειρά χρησιμοποιείται το μοντέλο που εκπαιδεύτηκε από την πρώτη χρονοσειρά του dataset κ.ο.κ.).
- Αν η παράμετρος -n που δόθηκε από τον χρήστη είναι μεγαλύτερη από το 20, τότε:
    - Για τις 20 πρώτες χρονοσειρές ακολουθείται η διαδικασία που εξηγήθηκε παραπάνω, δηλαδή γίνονται προβλέψεις χρησιμοποιώντας τα 2 αντίστοιχα μοντέλα.
    - Για τις υπόλοιπες χρονοσειρές χρησιμοποιείται μόνο το μοντέλο εκπαιδευμένο ανά σύνολο χρονοσειρών. Δηλαδή παρουσιάζεται μόνο ένα γράφημα.

Στην συνέχεια παρουσιάζουμε τις προβλέψεις του μοντέλου μας για την πρώτη χρονοσειρά του dataset.
Το πρώτο (πάνω) γράφημα αντιστοιχεί στο μοντέλο το οποίο έχει εκπαιδευτεί με ολόκληρο το σύνολο των χρονοσειρών. Η κόκκινη καμπύλη αντιστοιχεί στην αρχική καμπύλη (δηλαδή στο δεύτερο μισό της πρώτης καμπύλης του dataset) και η μπλε καμπύλη αντιστοιχεί στις προβλέψεις που έκανε το μοντέλο.
Το δεύτερο (κάτω) γράφημα αντιστοιχεί στο μοντέλο το οποίο έχει εκπαιδευτεί με μια μόνο χρονοσειρά. Συγκεκριμένα έχει εκπαιδευτεί με το πρώτο μισό της ίδιας χρονοσειράς. Η κόκκινη καμπύλη αντιστοιχεί στην αρχική καμπύλη (δηλαδή στο δεύτερο μισό της πρώτης καμπύλης του dataset) και η πράσινη καμπύλη αντιστοιχεί στις προβλέψεις που έκανε το μοντέλο.



Και στις δύο περιπτώσεις παρατηρούμε πολύ καλά αποτελέσματα.
Πιο συγκεκριμένα, τα curves που προκύπτουν από τις προβλέψεις των μοντέλων βρίσκονται πολύ κοντά στο αρχικό curve του dataset.
Ελέγχοντας τις επιμέρους τιμές των προβλέψεων παρατηρούμε πως όλες είναι πολύ κοντά στις τιμές της χρονοσειράς.
Από την παραπάνω εικόνα φαίνεται πως τα δύο μοντέλα έχουν ίδια απόδοση.
Ελέγχοντας τις επιμέρους τιμές των προβλέψεων και στις δύο περιπτώσεις παρατηρήσαμε πως το μοντέλο το οποίο έχει εκπαιδευτεί με το σύνολο των χρονοσειρών έχει ελάχιστα καλύτερη απόδοση. Δηλαδή οι τιμές των προβλέψεων έχουν λιγότερο από τις πραγματικές τιμές. Ωστόσο η διαφορά των δύο μοντέλων είναι μικρή.

Όπως έχει αναφερθεί και παραπάνω, το συγκεκριμένο “κομμάτι” της χρονοσειράς που παρουσιάστηκε δεν έχει χρησιμοποιηθεί για την εκπαίδευση του μοντέλου, δηλαδή είναι άγνωστη σε αυτό. Δεδομένου αυτού, η απόδοση που παρατηρούμε είναι πολύ καλή.

Την ίδια συμπεριφορά και απόδοση παρατηρούμε για οποιαδήποτε χρονοσειρά κάνουμε forecast. Αυτό επιβεβαιώνεται εύκολα εκτελώντας το πρόγραμμα και επιλέγοντας μεγάλη τιμή για τον αριθμό -n, έτσι ώστε να παρουσιαστούν πολλές forecasted χρονοσειρές.

Παραδοτέο αρχείο - forecast.py

Η διαδικασία που ακολουθείται κατα την εκτέλεση του παραδοτέου αρχείου είναι:

- Αρχικά διαβάζονται οι παράμετροι που έδωσε ο χρήστης από την γραμμή εντολων. Οι παράμετροι είναι:
    - -d <dataset> : και αντιστοιχεί στο path του dataset το οποίο θα χρησιμοποιηθεί για την Αξιολόγηση του μοντέλου.
    - -n <number of time series selected> : ο αριθμός των χρονοσειρών του dataset οι οποίες θα χρησιμοποιηθούν και θα παρουσιαστούν κατά την Αξιολόγηση του μοντέλου. Δηλαδή θα παρουσιαστεί το forecasting των n πρώτων χρονοσειρών του dataset.
- Χωρίζουμε το dataset σε train και test set όπως ακριβώς κάναμε και κατά την εκπαιδευση του μοντέλου, έτσι ώστε για την αξιολόγηση του μοντέλου να χρησιμοποιηθούν “κομμάτια” της χρονοσειράς τα οποία ΔΕΝ έχουν χρησιμοποιηθεί για την εκπαιδευση του μοντέλου. Για το train set παίρνουμε το πρώτο μισό κάθε χρονοσειράς ενώ για το test set παίρνουμε το δεύτερο μισό. Η διαδικασία εξηγείται αναλυτικά στην παράγραφο Διαχωρισμός των δεδομένων σε train και test set.
- Φορτώνεται το μοντέλο το οποίο έχει εκπαιδευτεί ανά σύνολο χρονοσειρών. *Παρατήρηση: αλλάζοντας την τιμή της μεταβλητής TRAIN_NEW_MODEL σε True, αντί να φορτωθεί το εκπαιδευμένο μοντέλο εκπαιδεύεται ένα νέο μοντέλο με βάση το dataset που έχει δοθεί χρησιμοποιώντας τις n πρώτες χρονοσειρές. Ο κώδικας που εκτελείται στην περίπτωση αυτή είναι αυτός που χρησιμοποιήσαμε για την εκπαιδευση του μοντέλου μας (με την διαφορά ότι εμείς χρησιμοποίησαμε όλες τις χρονοσειρές για την εκπαίδευση). Η τιμή της μεταβλητής TRAIN_NEW_MODEL είναι ορισμένη σε False, έτσι ώστε να φορτώνεται το ήδη εκπαιδευμένο μοντέλο κατά την εκτέλεση του προγράμματος.*
- Στην συνέχεια εκτελείται για κάθε ένα από τα **n** πρώτα curves του dataset η διαδικασία που περιγράφεται αναλυτικά στην παράγραφο Αξιολόγηση του μοντέλου.
  Επιγραμματικά, για κάθε μια χρονοσειρά:
    - Παίρνουμε το δεύτερο μισό της χρονοσειράς (test set) και δημιουργούμε μέσω αυτού τα αντίστοιχα επικαλυπτώμενα windows μήκους WINDOW_SIZE.
    - Εφαρμόζουμε scaling στο κάθε ένα window ξεχωριστά.
    - Δίνουμε σαν input στο μοντέλο μας το σύνολο των windows έτσι ώστε να κάνει predict την αντίστοιχη επόμενη τιμή για κάθε ένα window.
    - Κάνουμε unscale τα prediction που προέκυψαν, μέσω της inverse_transform.
    - Για τα πρώτα 20 curves (ή αν n<20 τότε για όλα τα curves):

        - Φορτώνουμε το αντίστοιχο μοντέλο το οποίο έχει γίνει train ανά χρονοσειρά και ακολουθούμε την ίδια διαδικασία με πριν έτσι ώστε να προκύψουν τα αντίστοιχα predictions και από αυτό το μοντέλο.
        - Παρουσιάζουμε δύο γραφικές παραστάσεις όπου αντιστοιχούν στις προβλέψεις του μοντέλου που εκπαιδεύτηκε ανά σύνολο χρονοσειρών και στο μοντέλο ανά χρονοσειρά
    - Αν n>20 τότε για τα υπόλοιπα curves παρουσιάζουμε σε ένα γράφημα τις προβλέψεις που προέκυψαν από το μοντέλο που εκπαιδεύτηκε ανα σύνολο χρονοσειρών.

Μέρος Β - Anomaly Detection

Στο δεύτερο μέρος της εργασίας υλοποιήσαμε ένα αναδρομικό νευρωνικό δίκτυο LSTM αυτοκωδικοποίησης χρονοσειρών το οποίο περιλαμβάνει στρώματα κωδικοποίησης και αποκωδικοποίησης.
Χρησιμοποιώντας το δίκτυο αυτό και ορίζοντας το κατάλληλο κατώφλι του μέσου απόλυτου σφάλματος επιτυγχάνουμε την ανίχνευση ανωμαλιών σε χρονοσειρές.

Διαχωρισμός των δεδομένων σε train και test set

Αρχικά χωρίζουμε τα δεδομένα μας σε train και test set.
Ο διαχωρισμός γίνεται με βάση το μήκος των χρονοσειρών, στο training set έχουμε συμπεριλάβει τις πρώτες 1825 τιμές από τις δεδομένες χρονοσειρές και στο test set έχουμε τις υπόλοιπες 1825 τιμές για τις ίδιες βεβαια χρονοσειρές (με λίγα λόγια έχουν χωριστεί στην μέση, 50-50).
Έχοντας ορίσει τα train και test set, χρησιμοποιούμε το train set για την εκπαίδευση του μοντέλου και το test set για να δοκιμάσουμε την απόδοση του μοντέλου μας έπειτα από το πέρας της εκπαιδευσης, δίνοντας του "άγνωστα" δεδομένα.

Επομένως το train set περιέχει τιμές από όλες χρονοσειρές και κατ' επέκταση το μοντέλο μας εκπαιδεύεται χρησιμοποιώντας όλες τις χρονοσειρές του dataset.
Αντίστοιχα το test set περιέχει τιμές από όλες χρονοσειρές και κατ' επέκταση δοκιμάζουμε το εκπαιδευμένο μοντέλο μας χρησιμοποιώντας "άγνωστες" τιμές από διαφορετικές χρονοσειρές οι οποίες έχουν διαφορετική συμπεριφορά μεταξύ τους.

Ορισμός υπερ-παραμέτρων

Όπως αναφέρεται και στην συνέχεια, για την εκπαιδευση του μοντέλου χρησιμοποιούμε ένα μικρό υποσύνολο του train set σαν validation set.
Μέσω του validation set μπορούμε να παρακολουθήσουμε την συμπεριφορά του μοντέλου μας σε ¨άγνωστα¨ δεδομένα κατα την διάρκεια εκπαίδευσης του και να το τροποποιήσουμε έτσι ώστε να αποδίδει εξίσου καλά και στο validation set.
Πιο συγκεκριμένα, παρακολουθώντας το training loss και το validation loss για κάθε ένα epoch, και κατασκευάζοντας τα αντίστοιχα loss vs epochs curves μπορούμε να παρατηρήσουμε την συμπεριφορά του μοντέλου κατά την εκπαίδευση και να τροποποιήσουμε τις υπερ-παραμέτρους του μοντέλου έτσι ώστε να αυξήσουμε την απόδοση του ενώ ταυτόχρονα να εξαλείψουμε την παρουσία overfit ή underfit στο μοντέλο μας.
Έτσι, έπειτα από τους αντίστοιχους πειραματισμούς, ορίζουμε τα παρακάτω hyperparameters:

- EPOCHES = 5, λόγω του μεγάλου πλήθους των δεδομένων του train set το μοντέλο δεν χρειάζεται πολλά epochs για να εκπαιδευτεί. Δοκιμάζοντας μεγαλύτερο αριθμό epochs η εκπαίδευση του μοντέλου καθυστερούσε υπερβολικά πολύ ενώ ταυτόχρονα το train loss παρέμενε σταθερό (ή είχε αμελητέα μείωση) και το validation loss ξεκινούσε να αυξάνεται αντί να μειώνεται. Στον βέλτιστο αριθμό epochs καταλήξαμε χρησιμοποιώντας την τεχνική Early stopping κάνοντας monitor το validation loss και σταματώντας την εκπαιδευση όταν αυτό αρχίσει να αυξάνεται. Έτσι αποφευγουμε το overfit στο μοντέλο μας.
- BATCH_SIZE = 2048, γνωρίζουμε πως όσο μεγαλύτερο είναι το batch size τόσο πιο λίγο θόρυβο προκαλεί και τόσο καλύτερη γίνεται η εκτίμηση. Δοκιμάζοντας πιο μικρά μεγέθη (συνήθως δυνάμεις του 2) παρατηρήσαμε πως η εκπαίδευση καθυστερούσε αρκετά και τα αποτελέσματα δεν ήταν ικανοποιητικά, ενώ για μεγαλύτερα δεν παρατηρήθηκαν διαφορές στην απόδοση ωστόσο αυξανόταν ο αριθμός των epochs που χρειάζοντας έτσι ώστε το μοντέλο να έχει την επιθυμητή απόδοση.
- LEARNING_RATE = 0.001, δοκιμάσαμε μεγαλύτερες τιμές, στις οποίες το μοντέλο λόγω της πιο γρήγορης εκμάθησης κατα την διάρκεια της εκπαίδευσης παρουσίαζε αστάθεια στα αποτελέσματα και πολλές φορές το μοντέλο παρουσίαζε overfit. Τα παραπάνω επιβεβαιώνονται παρακολουθώντας την πορεία των train και validation loss curves. Δοκιμάζοντας μικρότερες τιμές, το μοντέλο χρειαζόταν περισσότερα epochs για να εκπαιδευτεί και τελικά δεν είχε την επιθυμητή απόδοση..
- WINDOW SIZE = 60, η οποία παράμετρος αφορά ουσιαστικά τα time steps στα οποία θα χωριστούν οι μετοχές του train set, ώστε στη συνέχεια να φτιάξουμε τα κατάλληλα σύνολα (X_train και y_train) που θα χρησιμοποιηθούν για την εκπαίδευση του μοντέλου. Δηλαδή το input του μοντέλου μας θα είναι 60 συνεχόμενες τιμες μιας χρονοσειράς. Δοκιμάσαμε τιμές από 10 έως 70, χωρίς να παρατηρούμε μεγάλες διαφορές, τελικά καταλήξαμε να βάλουμε την τιμή 60 που μας προέκυπταν τα καλύτερα αποτελέσματα.
  Περισσότερες λεπτομέρειες για το “σπάσιμο” της χρονοσειράς σε windows εξηγούνται στην συνέχεια.

Προ-επεξεργασία δεδομένων - Scaling

Στη συνέχεια, κατασκευάζουμε την δομή που θα χρησιμοποιηθεί για την εκπαίδευση του μοντέλου (δηλαδή τα σύνολα X_train και y_train) εφαρμόζοντας ταυτόχρονα και το απαραίτητο scaling (ώστε να κανονικοποιήσουμε τις τιμές των μετοχών).
Παρατήρηση: *Η διαδικασία είναι αντίστοιχη με αυτήν του πρώτου μέρους, επομένως αρκετά πράγματα επαναλαμβάνονται.*
Όπως έχει αναφερθεί και παραπάνω το train set (και κατ' επέκταση τα X_train και y_train) κατασκευάζεται από το πρώτο μισό της κάθε χρονοσειράς. Έτσι στην παράγραφο αυτή, όταν αναφερόμαστε σε μία χρονοσειρά εννοούμε στο πρώτο μισό της χρονοσειράς.
Για την παραγωγή του X_train οι χρονοσειρές χωρίζονται σε διαδοχικά επικαλυπτώμενα sets μήκους ίσου με το WINDOW_SIZE (δηλαδή 60). Πιο συγκεκριμένα το πρώτο set τιμών αποτελείται από τις πρώτες window τιμές της χρονοσειράς, στο δεύτερο set έχει αφαιρεθεί η πρώτη τιμή του πρώτου set και έχει προστεθεί η window+1 τιμή της χρονοσειράς κ.ο.κ.. Το τελευταίο set περιέχει window τιμές με την τελευταία τιμή του set να αντιστοιχεί στην προ-τελευταία τιμή της χρονοσειράς. Δηλαδή τα windows είναι επικαλυπτώμενα.
Σε κάθε ένα set window τιμών του X_train αντιστοιχεί μια τιμή στο y_train. Η τιμή αυτή είναι η αμέσως επόμενη τιμή της χρονοσειράς.
Έτσι, τελικά το X_train αποτελείται από sets όλων των χρονοσειρών και το y_train αντίστοιχα. Όπως είναι προφανές, κανένα set δεν αποτελείται από τιμές δύο διαφορετικών χρονοσειρών.

Σημαντική παρατήρηση

Η επιλογή το y_train να αντιστοιχεί στην αμέσως επόμενη τιμή του αντιστοιχου window (σε συνδιασμό το το TimeDistributed layer στο δίκτυο μας), έγινε με βάση το tutorial που μας δόθηκε καθώς ακολουθείται η ίδια προσέγγιση.
Με βάση την θεωρία, καθώς το μοντέλο μας είναι ένας auto-encoder, το y_train θα έπρεπε να είναι το ίδιο με το X_train αφού το τελικό output ενός αποκωδικοποιητή θέλουμε να είναι όσο το δυνατόν πιο κοντά στο input.
Ωστόσο εφαρμόζοντας και τις δύο προσεγγίσεις παρατηρήσαμε καλύτερα αποτελέσματα με την πρώτη, δηλαδή το y_train να αντιστοιχεί στην αμέσως επόμενη τιμή.
Έπειτα από συζήτηση για το συγκεκριμένο θέμα στο φροντιστήριο με τον κ. Χαμόδρακα καταλήξαμε στο ότι και οι δύο λύσεις είναι αποδεκτές. Έτσι επιλέξαμε αυτήν την προσέγγιστη καθώς μας προέκυπταν καλύτερα αποτελέσματα.

Για το scaling χρησιμοποιούμε τον StandardScaler, επιπλέον δοκιμάσαμε και τον MinMaxScaler, όμως η απόδοση του μοντέλου μας δεν ήταν τόσο ικανοποιητική.

Όσον αφορά το scaling, εφαρμόζουμε ξεχωριστό scaling σε κάθε ένα window (συγκεκριμένα ανά 60 time steps). Δηλαδή ο scaler γίνεται fit σε κάθε ένα window ξεχωριστά. Το εύρος τιμών εσωτερικά ενός window είναι σχετικά μικρό. Έτσι οι scaled τιμές που προκύπτουν για κάθε ένα window είναι ομοιόμορφες σε κάθε περίπτωση. Πρόσθετα, πειραματιστήκαμε κάνοντας και scale ανά μετοχή, συγκρίνοντας τα αποτελέσματα του μοντέλου με το scale ανα window, στο οποίο τελικά καταλήξαμε αφού η απόδοση του μοντέλου ήταν αρκετά καλύτερη. Το scale ανά μετοχή δεν είναι τόσο αποδοτικό, διότι η αναπαράσταση που προκύπτει δεν αντιπροσωπεύει ακριβώς τις αρχικές τιμές της κάθε μετοχής, καθώς ορισμένες μετοχές έχουν μεγάλο εύρος τιμών ενώ άλλες έχουν πολύ μικρό με αποτέλεσμα οι scaled τιμές που προκύπτουν να μην είναι ομοιόμορφες μεταξύ των διαφορετικών χρονοσειρών.

Ορισμός του αναδρομικού νευρωνικού δικτύου αποκωδικοποίησης

Το αναδρομικό νευρωνικό δίκτυο αποτελείται από τον κωδικοποιητή (encoder) και τον αποκωδικοποιητή (decoder).
Πιο συγκεκριμένα, στο νευρωνικό δίκτυο που υλοποιήσαμε ο encoder αποτελείται από:

- Ένα LSTM layer με 128 units
- Ένα LSTM layer με 64 units

Αντίστοιχα ο decoder αποτελείται από:

- Ένα LSTM layer με 64 units
- Ένα LSTM layer με 128 units

Δηλαδή, τόσο ο encoder όσο και ο decoder αποτελούνται από δύο στρώματα LSTM. Στον decoder ο αριθμός των units μειώνεται κατά την μετάβαση από το πρώτο layer στο δεύτερο, ενώ αντίθετα στον decoder ο αριθμός των units αυξάνεται.
Τα layers αυτά προσθέτουν την επιθυμητή πολυπλοκότητα στο μοντέλο η οποία σε συνδυασμό με τον μεγάλο όγκο των δεδομένων εκπαίδευσης αλλά και την επιλογή των κατάλληλων παραμέτρων έχει ως αποτέλεσμα την πολύ καλή απόδοση του τελικού μοντέλου.
Δοκιμάζοντας να κάνουμε πιο “περίπλοκο” το μοντέλο μας προσθέτοντας επιπλέον layers ή αυξάνοντας τον αριθμό των units η απόδοση του μοντέλου αυξανόταν μόνο για το train set και όχι για το validation set. Δηλαδή, παρουσίαζε overfit.
Δοκιμάζοντας απλούστερα μοντέλα, δεν είχαν την επιθυμητή απόδοση σε σχέση με αυτό που επιλέξαμε τελικά.

Έπειτα από το τελευταίο LSTM layer του decoder ακολουθεί ένα Dense layer το οποίο εφαρμόζεται σε κάθε slice μέσω ενός TimeDistributed layer και έτσι επαναφέρουμε τα

δεδομένα στην αρχική τους διάσταση (δηλαδή ίση με το window_size). Έτσι προκύπτει το τελικό output του auto encoder το οποίο αντιστοιχεί στο αρχικό αποκωδικοποίηση του αρχικού input.

Τέλος, ενδιάμεσα του encoder και του decoder υπάρχει ένα RepeatVector layer. Πιο συγκεκριμένα, το output του encoder είναι 64 τιμές, όσος δηλαδή και ο αριθμός των units του δεύτερου LSTM layer του. Το RepeatVector layer επαναλαμβάνει το output αυτό window φορές έτσι ώστε να δοθεί σαν input στον decoder. Δηλαδή παράγει ένα tensor μεγέθους (60,64) το οποίο δίνεται σαν είσοδος στο πρώτο LSTM layer του decoder.

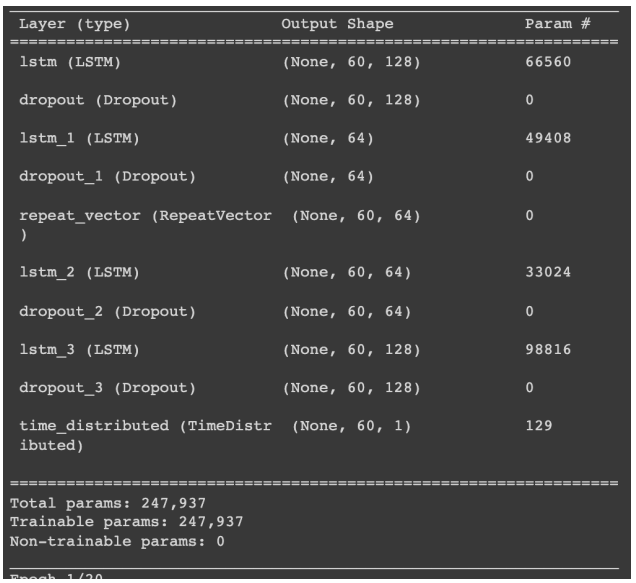
Επίσης, έτσι ώστε να αυξήσουμε την απόδοση του μοντέλου αλλά και να εξαλείψουμε τελείως το overfit προσθέσαμε:

- Dropout Layers (με dropout rate = 0.2) έπειτα από κάθε LSTM layer έτσι ώστε να αποτρέψουμε το μοντέλο από το να παρουσιάσει overfitting.
- ReLU activation function σε όλα τα LSTM layer, καθώς παρατηρήσαμε βελτίωση στην απόδοση έπειτα από την προσθήκη τους.

Δηλαδή, συνολικά, το νευρωνικού δικτύου αποκωδικοποίησης αποτελείται από τα εξής:

- LSTM layer με 128 units και relu activation function
- Dropout layer με dropout rate = 0.2
- LSTM layer με 64 units και relu activation function
- Dropout layer με dropout rate = 0.2
- RepeatVector layer με n=WINDOW_SIZE
- LSTM layer με 64 units και relu activation function
- Dropout layer με dropout rate = 0.2
- LSTM layer με 128 units και relu activation function
- Dropout layer με dropout rate = 0.2
- Dense layer το οποίο εφαρμόζεται μέσω ενός Time Distributed layer

Αναλυτικότερα η εικόνα του μοντέλου, όπως παρουσιάζεται από την συνάρτηση model.summary(), είναι η εξής:



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 60, 128)           66560

dropout (Dropout)            (None, 60, 128)           0

lstm_1 (LSTM)                (None, 64)                49408

dropout_1 (Dropout)          (None, 64)                0

repeat_vector (RepeatVector  (None, 60, 64)            0
)

lstm_2 (LSTM)                (None, 60, 64)            33024

dropout_2 (Dropout)          (None, 60, 64)            0

lstm_3 (LSTM)                (None, 60, 128)           98816

dropout_3 (Dropout)          (None, 60, 128)           0

time_distributed (TimeDistr  (None, 60, 1)             129
ibuted)

=================================================================
Total params: 247,937
Trainable params: 247,937
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
```

Εκπαίδευση του μοντέλου

Όσον αφορά την εκπαιδευση του μοντέλου αρχικά ορίζουμε τον optimizer καθώς και το loss function που θα χρησιμοποιηθούν. Πιο συγκεκριμένα, ορίζουμε:

- Optimizer : Adam , δίνοντας του σαν παράμετρο το learning rate το οποίο ορίσαμε παραπάνω.
- Loss Function : Mean Absolute Error

Επίσης κατά την κλήση της συνάρτησης fit (η οποία πραγματοποιεί την εκπαίδευση), ορίζουμε τις εξής παραμέτρους:

- X_train και y_train. Εξηγούνται στη παράγραφο Προ-επεξεργασία δεδομένων - Scaling.
- epochs και batch_size τα οποία έχουν οριστεί στον Ορισμός υπερ-παραμέτρων.
- validation_split=0.1 , δηλαδή ορίζουμε το 10% του train set να θεωρηθεί σαν validation set έτσι ώστε να παρακολουθούμε την πορεία του μοντέλου κατα την εκπαιδευση του πάνω σε άγνωστα δεδομένα. Έτσι, καταφέρνουμε να γενικεύσουμε το μοντέλο μας, δηλαδή να μην παρουσιάζει overfit ή underfit.
- Early stopping με βάση το validation loss και patience = 3. Πιο συγκεκριμένα παρακολουθείται το validation loss και σε περίπτωση που το validation loss αυξάνεται σε τρία διαδοχικά epochs τότε η εκπαίδευση σταματά καθώς το μοντέλο δεν βελτιώνει πλέον την απόδοση του σε άγνωστα δεδομένα, δηλαδή αρχίζει να παρουσιάζει overfit.

Εκπαιδευμένο μοντέλο

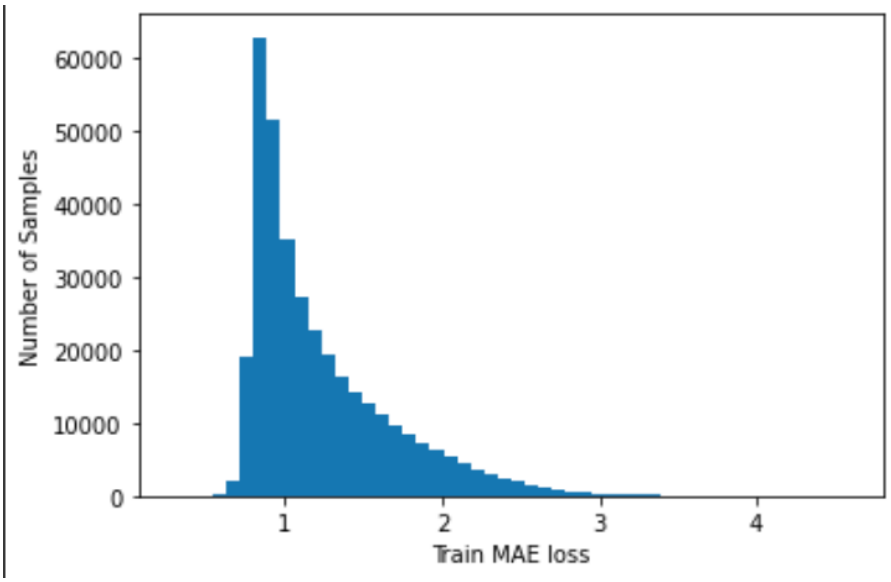
Με βάση τα παραπάνω έχουμε εκπαιδεύσει ένα μοντέλο χρησιμοποιώντας τις πρώτες 200 χρονοσειρές του dataset που μας δόθηκαν.
Η επιλογή να εκπαιδεύσουμε το μοντέλο χρησιμοποιώντας μόνο 200 χρονοσειρές έγινε καθώς:

- Χρησιμοποιώντας όλες τις χρονοσειρές η εκπαίδευση είναι πολύ χρονοβόρα και διαρκεί αρκετές ώρες.
- Ταυτόχρονα η προσθήκη περισσότερων χρονοσειρών δεν αυξάνει την απόδοση του μοντέλου καθώς οι 200 είναι αρκετές για να εκπαιδευτεί πλήρως το μοντέλο μας.

Το μοντέλο αυτό το έχουμε αποθηκεύσει στο αντίστοιχο αρχείο models/part2/part2_model.h5 και χρησιμοποιείται από το εκτελέσιμο αρχείο του δεύτερου μέρους της εργασίας (όπως εξηγείται και στην συνέχεια).

Επιλογή του Threshold

Έχοντας το εκπαιδευμένο μοντέλο εκτελούμε την συνάρτηση predict δίνοντας σαν είσοδο στο μοντέλο το σύνολο X_train, δηλαδή το σύνολο με το οποίο εκπαιδεύτηκε.
Με βάση το output που προκύπτει υπολογίζουμε για κάθε ένα window το μέσο απόλυτο σφάλμα, το οποίο προκύπτει από την διαφορά των πραγματικών τιμών των μετοχών από τις τιμές που προέκυψαν από την αποκωδικοποίηση.
Έχοντας υπολογίσει το μέσο απόλυτο σφάλμα για κάθε ένα input που δόθηκε στο μοντέλο παρουσιάζουμε γραφικά τα losses που προέκυψαν σε μορφή ιστογράμματος.
Το ιστόγραμμα που προκύπτει είναι το εξής:



Παρατηρούμε ότι:

- Το loss κυμαίνεται κατά βάση στο διάστημα [0.7 , 1.5]
- Για loss>1, όσο αυξάνεται το loss τόσο μειώνεται και η συχνότητα του.
- Για loss>2.5, η συχνότητα είναι πάρα πολύ μικρή.

Έτσι, επιλέγουμε σαν κατώφλι του μέσου απόλυτου σφάλματος την τιμή 2.25 καθώς από το σημείο αυτό και μετά η συχνότητα είναι πολύ μικρή, δηλαδή έχουμε μια ανωμαλία.
Επομένως ορίζουμε: **THRESHOLD = 2.25**.
Αυξάνοντας την τιμή του threshold παρουσιάζονται λιγότερες ανωμαλίες καθώς είμαστε πιο "ανεκτικοί" όσον αφορά το loss.
Αντίθετα μειώνοντας την τιμή του threshold, περισσότερα σημεία θεωρούνται ως ανωμαλίες.

Παρατήρηση σχετικά με το threshold

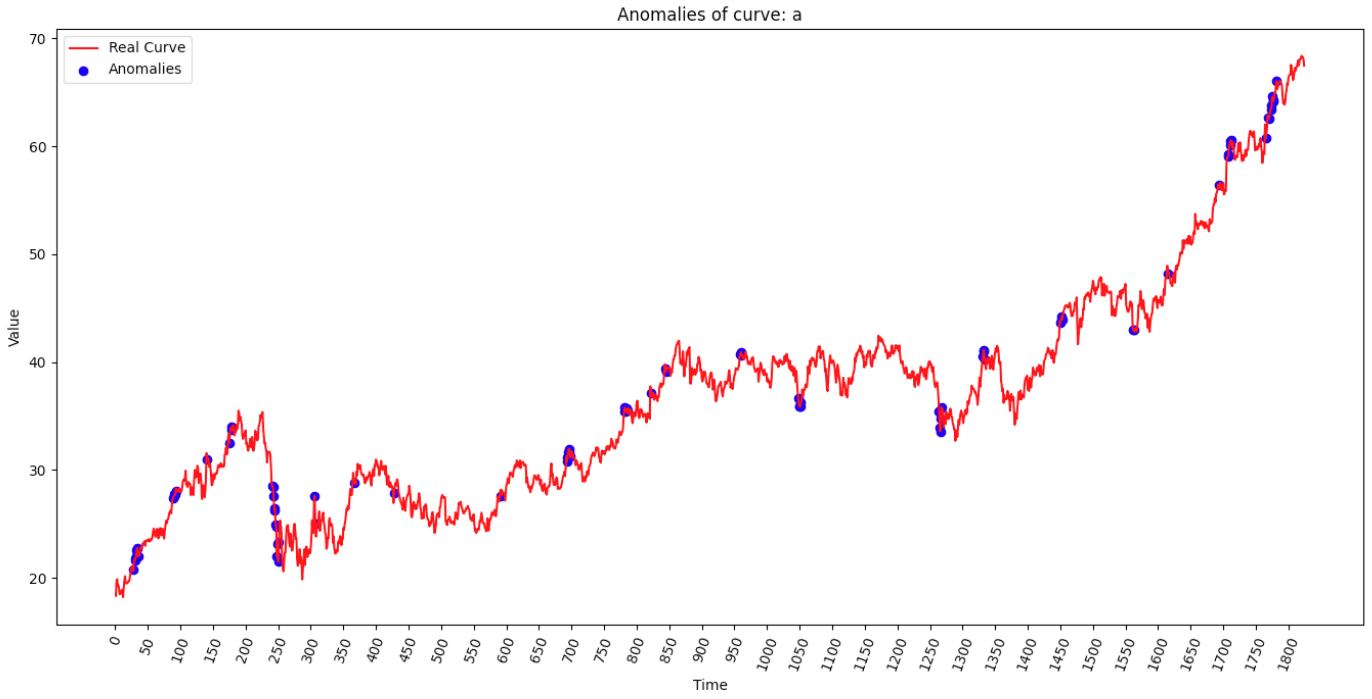
Παρατηρούμε σχετικά υψηλές τιμές τα mean absolute errors που προκύπτουν και κατ' επέκταση στην τιμή του threshold που επιλέξαμε.
Οι υψηλές αυτές τιμές οφείλονται στην επιλογή να εφαρμόσουμε το scaling ξεχωριστά σε κάθε window (όπως αναλύθηκε παραπάνω).
Δοκιμάζοντας να εφαρμόσουμε το scaling σε ολόκληρη την χρονοσειρά παρατηρούμε μικρότερες τιμές για τα mean absolute errors (και κατ' επέκταση για το βέλτιστο threshold). Πιο συγκεκριμένα κυμαίνονται στο διάστημα [0,1] με το μέσο απόλυτο σφάλμα να είναι 0.3. Ωστόσο το μοντέλο παρουσιάζει απροσδιόριστη συμπεριφορά κατά την εύρεση των ανωμαλιών στο test set. Δηλαδή, σε ορισμένες χρονοσειρές προκύπτει υπερβολικά μεγάλο πλήθος ανωμαλιών (χωρίς στην πραγματικότητα όλα να αποτελούν ανωμαλίες) ενώ ταυτόχρονα σε άλλες χρονοσειρές του test set δεν εντοπίζει καμία ανωμαλία ενώ θα έπρεπε. Αντίθετα, με την επιλογή μας να εφαρμόσουμε scaling ανα window το μοντέλο μας έχει πολύ καλή συμπεριφορά, εντοπίζοντας σωστά τις ανωμαλίες σε όλες τις χρονοσειρές του test set.

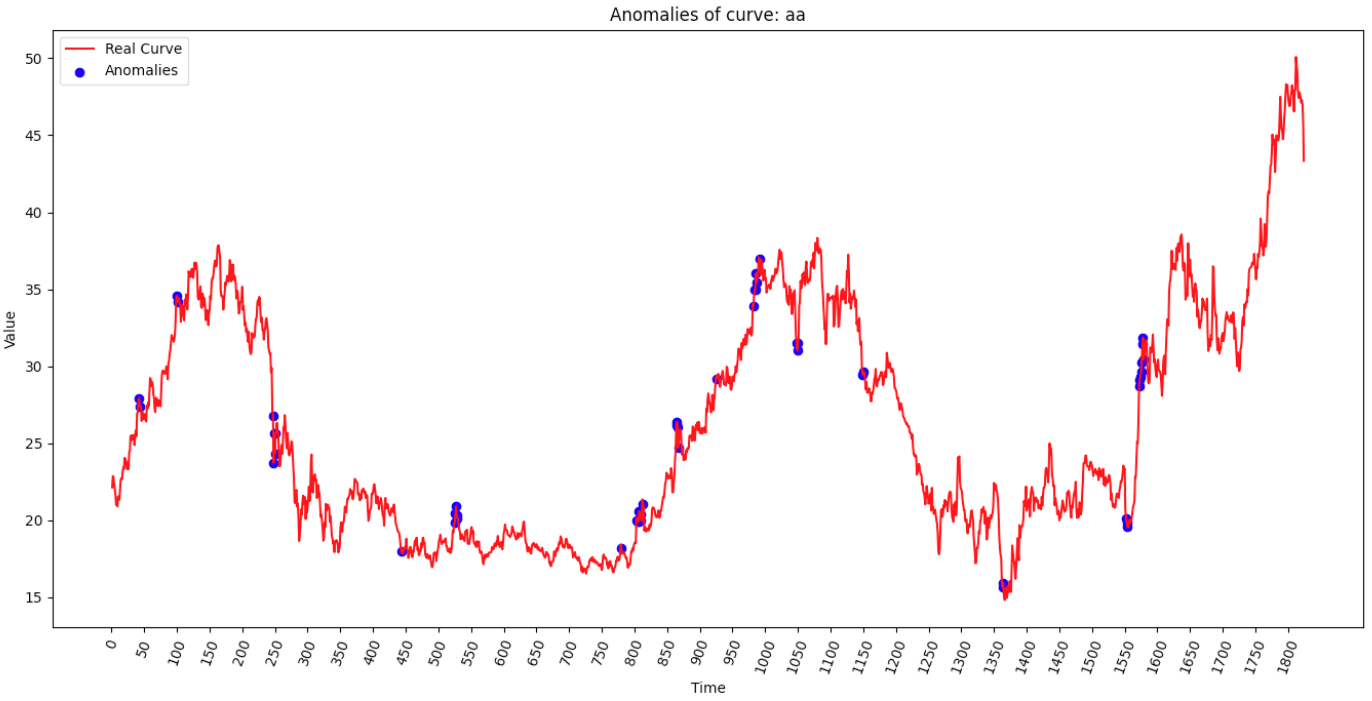
Αξιολόγηση του μοντέλου

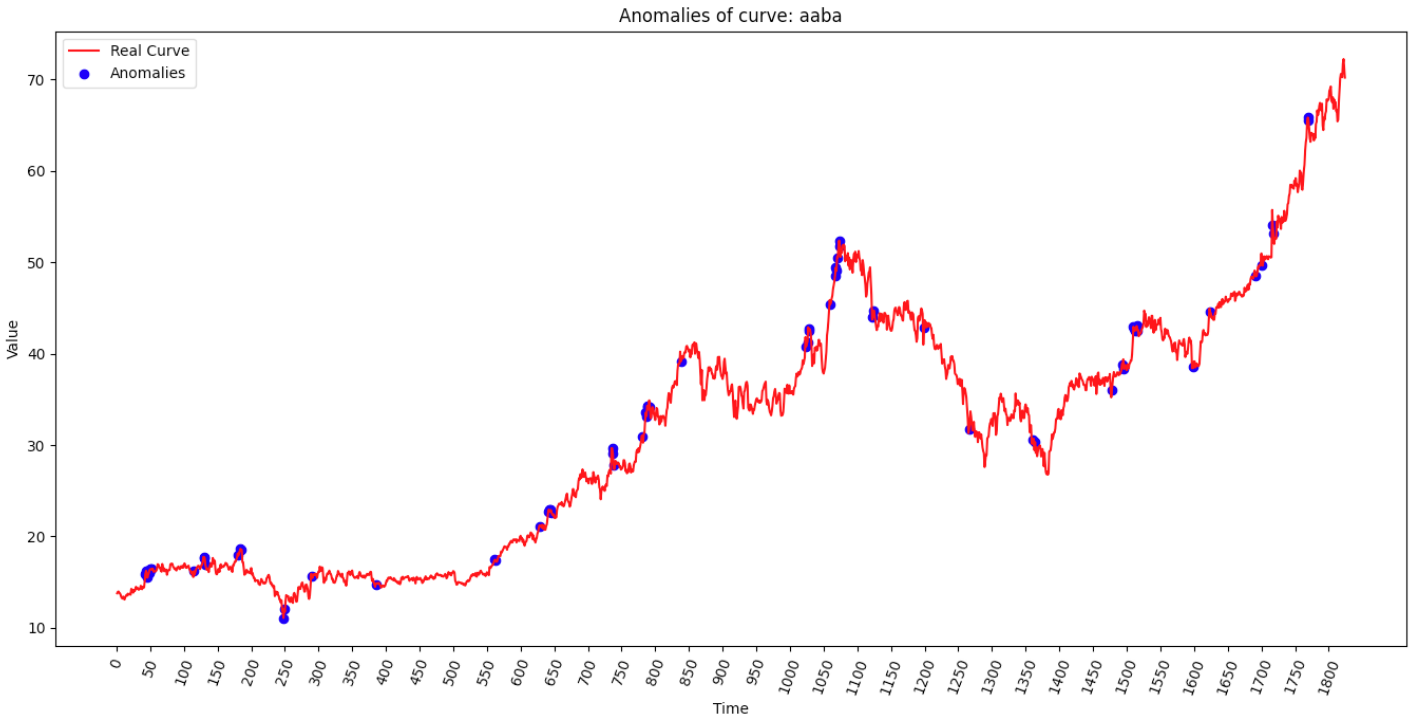
Όπως αναφέρθηκε και παραπάνω η εκπαίδευση το μοντέλου γίνεται χρησιμοποιώντας το πρώτο μισό του κάθε curve, δηλαδή τις πρώτες 1825 τιμές. Έχοντας το εκπαιδευμένο μοντέλο και ανάλογα με την παράμετρο -n την οποία έχει ορίσει ο χρήστης κατά την εκτέλεση του προγράμματος εκτελείται η ακόλουθη διαδικασία για την αξιολόγηση του μοντέλου.
Για κάθε ένα μια τις n πρώτες χρονοσειρές του dataset:

- Χρησιμοποιώντας το δεύτερο μισό της χρονοσειράς, δηλαδή τις 1825 τελευταίες τιμές της δημιουργούμε τα σύνολο X_test . Το σύνολο αυτό δημιουργείται ακριβώς με τον ίδιο τρόπο που δημιουργείται το σύνολο X_train και εξηγείται αναλυτικά στην παράγραφο Προ-επεξεργασία δεδομένων - Scaling. Δηλαδή δημιουργούμε επικαλυπτώμενα sets μήκους 60 (window_size) από διαδοχικές τιμές της χρονοσειράς. Όπως και στο train set, εφαρμόζουμε scaling ανά window.
- Έτσι το σύνολο X_test αντιστοιχεί στο input που θα δοθεί στο μοντέλο μας έτσι ώστε να δοκιμάσουμε την απόδοση του. Το X_test αποτελείται από 60αδες διαδοχικών τιμών στις οποίες έχει εφαρμοστεί (ξεχωριστά σε κάθε μια) το αντίστοιχο scaling. Για κάθε μια 60αδα το μοντέλο μας παράγει σαν output μια αντίστοιχη αποκωδικοποιημένη 60άδα.
- Εκτελώντας την συνάρτηση predict στο μοντέλο μας και δίνοντας σαν είσοδο το X_test, λαμβάνουμε σαν output τόσα sets (windows) όσος και ο αριθμός των windows που περιέχονται στο X_test.
- Στην συνέχεια χρησιμοποιώντας τις scaled τιμές που δώσαμε σαν input στο μοντέλο (δηλαδή το X_train) καθώς και τις τιμές που προέκυψαν σαν output του μοντέλου υπολογίζουμε για κάθε ένα window το μέσο απόλυτο σφάλμα
- Με βάση τα losses που προέκυψαν, εντοπίζουμε ποια από αυτά είναι μεγαλύτερα από το threshold το οποίο έχουμε ορίσει και έτσι προκύπτουν οι ανωμαλίες. Πιο συγκεκριμένα, αν το loss ενός window είναι μεγαλύτερο από το threshold τότε η αμέσως επόμενη τιμή της χρονοσειράς θεωρείται μια ανωμαλία.
- Τέλος, έχοντας εντοπίσει όλα τα σημεία στα οποία παρουσιάζεται μια ανωμαλία, παρουσιάζουμε γραφικά την αρχική καμπύλη του dataset και τα σημεία πάνω σε αυτήν στα οποία παρουσιάζεται μια ανωμαλία. Η καμπύλη που παρουσιάζεται αντιστοιχεί στο δεύτερο μισό της αρχικής καμπύλης δηλαδή στην καμπύλη του test set.

Στην συνέχεια παρουσιάζουμε Τα αποτελέσματα που προκύπτουν για τις 3 πρώτες καμπύλες του dataset.
Η κόκκινη καμπύλη αντιστοιχεί στην αρχική καμπύλη (δηλαδή στο δεύτερο μισό της πρώτης καμπύλης του dataset)
Τα μπλε σημεία αντιστοιχούν στις ανωμαλίες που εντοπίστηκαν.


Anomalies of curve: a
Real Curve
Anomalies
Value
Time


Anomalies of curve: aa
Real Curve
Anomalies
Value
Time



Και στις τρεις περιπτώσεις παρατηρούμε πολύ καλά αποτελέσματα.
Πιο συγκεκριμένα, οι ανωμαλίες φαίνεται να προκύπτουν σε σημεία στα οποία η χρονοσειρά παρουσιάζει απότομη αύξηση ή μείωση σε σχέση με τις αμέσως προηγούμενες τιμές της.
Αυτό είναι και το αναμενόμενο και επιβεβαιώνει την καλή συμπεριφορά του μοντέλου μας.

Όπως έχει αναφερθεί και παραπάνω, το συγκεκριμένο “κομμάτι” της χρονοσειράς που παρουσιάστηκε δεν έχει χρησιμοποιηθεί για την εκπαίδευση του μοντέλου, δηλαδή είναι άγνωστη σε αυτό. Δεδομένου αυτού, η απόδοση που παρατηρούμε είναι πολύ καλή.

Αντίστοιχη συμπεριφορά παρατηρούμε για οποιαδήποτε χρονοσειρά. Αυτό επιβεβαιώνεται εύκολα εκτελώντας το πρόγραμμα και επιλέγοντας μεγάλη τιμή για τον αριθμό -n, έτσι ώστε να παρουσιαστούν πολλές χρονοσειρές με τις αντίστοιχες ανωμαλίες τους.

Παραδοτέο αρχείο - detect.py

Η διαδικασία που ακολουθείται κατα την εκτέλεση του παραδοτέου αρχείου είναι:

- Αρχικά διαβάζονται οι παράμετροι που έδωσε ο χρήστης από την γραμμή εντολων. Οι παράμετροι είναι:
    - -d <dataset> : και αντιστοιχεί στο path του dataset το οποίο θα χρησιμοποιηθεί για την Αξιολόγηση του μοντέλου.
    - -n <number of time series selected> : ο αριθμός των χρονοσειρών του dataset οι οποίες θα χρησιμοποιηθούν και θα παρουσιαστούν κατά την Αξιολόγηση του μοντέλου. Δηλαδή θα παρουσιαστουν οι ανωμαλίες των n πρώτων χρονοσειρών του dataset.
    - -mae <error value as double> : η τιμή του threshold με βάση την οποία προκύπτουν οι ανωμαλίες. Σε περίπτωση που δεν δοθεί αυτή η παράμετρος, ορίζεται σαν default τιμή η βέλτιστη τιμή για το threshold που επιλέξαμε βάση και παρουσιάσαμε στην παράγραφο Επιλογή του Threshold.
    - -print_mae : επιπλέον προαιρετική παράμετρος που προσθέσαμε έτσι ώστε να μπορεί να εκτυπωθεί το ιστόγραμμα των mae. Βλέπε στο τέλος της παραγράφου για περισσότερες λεπτομέρειες.
- Χωρίζουμε το dataset σε train και test set όπως ακριβώς κάναμε και κατά την εκπαιδευση του μοντέλου, έτσι ώστε για την αξιολόγηση του μοντέλου να χρησιμοποιηθούν “κομμάτια” της χρονοσειράς τα οποία ΔΕΝ έχουν χρησιμοποιηθεί για την εκπαιδευση του μοντέλου. Για το train set παίρνουμε το πρώτο μισό κάθε χρονοσειράς ενώ για το test set παίρνουμε το δεύτερο μισό. Η διαδικασία εξηγείται αναλυτικά στην παράγραφο Διαχωρισμός των δεδομένων σε train και test set.
- Φορτώνεται το μοντέλο το οποίο έχει εκπαιδευτεί ανά σύνολο χρονοσειρών. *Παρατήρηση: αλλάζοντας την τιμή της μεταβλητής TRAIN_NEW_MODEL σε True, αντί να φορτωθεί το εκπαιδευμένο μοντέλο εκπαιδεύεται ένα νέο μοντέλο με βάση το dataset που έχει δοθεί χρησιμοποιώντας τις n πρώτες χρονοσειρές. Ο κώδικας που εκτελείται στην περίπτωση αυτή είναι αυτός που χρησιμοποιήσαμε για την εκπαιδευση του μοντέλου μας (με την διαφορά ότι εμείς χρησιμοποίησαμε όλες τις χρονοσειρές για την εκπαίδευση). Η τιμή της μεταβλητής TRAIN_NEW_MODEL είναι ορισμένη σε False, έτσι ώστε να φορτώνεται το ήδη εκπαιδευμένο μοντέλο κατά την εκτέλεση του προγράμματος.*
- Στην συνέχεια εκτελείται για κάθε ένα από τα **n** πρώτα curves του dataset η διαδικασία που περιγράφεται αναλυτικά στην παράγραφο Αξιολόγηση του μοντέλου.
  Επιγραμματικά, για κάθε μια χρονοσειρά:

- Παίρνουμε το δεύτερο μισό της χρονοσειράς (test set) και δημιουργούμε μέσω αυτού τα αντίστοιχα επικαλυπτώμενα windows μήκους WINDOW_SIZE.
- Εφαρμόζουμε scaling στο κάθε ένα window ξεχωριστά.
- Δίνουμε σαν input στο μοντέλο μας το σύνολο των windows έτσι ώστε να μας επιστρέψει τα αντίστοιχα αποκωδικοποιημένα windows.
- Για κάθε ένα αποκωδικοποιημένο window υπολογίζουμε το μέσο απόλυτο σφάλμα με βάση το αρχικό window.
- Με βάση τα σφάλματα που θα προκύψουν και χρησιμοποιώντας το threshold που έχει οριστεί εντοπίζουμε τα σημεία της χρονοσειράς στα οποία παρουσιάζεται μια ανωμαλία.
- Τέλος παρουσιάζουμε γραφικά την αρχική test χρονοσειρά (δηλαδή το δεύτερο μισό της χρονοσειράς του dataset) και πάνω σε αυτήν παρουσιάζουμε τα σημεία στα οποία εντοπίστηκε μια ανωμαλία.

Επίσης υπάρχει η δυνατότητα να εκτυπωθεί το ιστόγραμμα που παρουσιάζει τα απόλυτα σφάλματα τα οποία προέκυψαν για το train set, δηλαδή όπως το ιστόγραμμα που παρουσιάζεται στην παράγραφο Επιλογή του Threshold.
Αυτό ενδεχομένως να φανεί χρήσιμο για την δοκιμή κάποιου νέου dataset, εκτός από αυτού που μας έχει δοθεί, έτσι ώστε να επιλέξουμε την βέλτιστη τιμή για το threshold.
Η δυνατότητα αυτή ενεργοποιείται εκτελώντας το πρόγραμμα με την επιπλέον παράμετρο -**print_mae** .

Μέρος Γ - Time Series Compression

Στο τρίτο μέρος της εργασίας κατασκευάσαμε ένα συνελικτικό νευρωνικό δίκτυο αποκωδικοποίησης χρονοσειρών το οποίο χρησιμοποιούμε για την μείωση της πολυπλοκότητας χρονοσειρών.

Διαχωρισμός των δεδομένων σε train και test set

Αρχικά χωρίζουμε τα δεδομένα μας σε train και test set.
Ο διαχωρισμός γίνεται με βάση το μήκος των χρονοσειρών, στο training set έχουμε συμπεριλάβει τις πρώτες 95% των τιμών από τις δεδομένες χρονοσειρές και στο test set έχουμε το υπόλοιπο 5% των τιμών για τις ίδιες βεβαια χρονοσειρές (με λίγα λόγια έχουν χωριστεί στην σε ποσοστό, 95-5).
Έχοντας ορίσει τα train και test set, χρησιμοποιούμε το train set για την εκπαίδευση του μοντέλου και το test set για να δοκιμάσουμε την απόδοση του μοντέλου μας έπειτα από το πέρας της εκπαιδευσης, δίνοντας του “άγνωστα” δεδομένα.

Επομένως το train set περιέχει τιμές από όλες χρονοσειρές και κατ’ επέκταση το μοντέλο μας εκπαιδεύεται χρησιμοποιώντας όλες τις χρονοσειρές του dataset.
Αντίστοιχα το test set περιέχει τιμές από όλες χρονοσειρές και κατ’ επέκταση δοκιμάζουμε το εκπαιδευμένο μοντέλο μας χρησιμοποιώντας “άγνωστες” τιμές από διαφορετικές χρονοσειρές οι οποίες έχουν διαφορετική συμπεριφορά μεταξύ τους.

Ορισμός υπερ-παραμέτρων

Για την εκπαιδευση του μοντέλου χρησιμοποιούμε το test set, που ορίστηκε στην αμέσως παραπάνω παράγραφο, σαν validation set.
Μέσω του validation set μπορούμε να παρακολουθήσουμε την συμπεριφορά του μοντέλου μας σε ¨άγνωστα¨ δεδομένα κατα την διάρκεια εκπαίδευσης του και να το τροποποιήσουμε έτσι ώστε να αποδίδει εξίσου καλά και στο validation set.
Πιο συγκεκριμένα, παρακολουθώντας το training loss και το validation loss για κάθε ένα epoch, και κατασκευάζοντας τα αντίστοιχα loss vs epochs curves μπορούμε να παρατηρήσουμε την συμπεριφορά του μοντέλου κατά την εκπαίδευση και να τροποποιήσουμε τις υπερ-παραμέτρους του μοντέλου έτσι ώστε να αυξήσουμε την απόδοση του ενώ ταυτόχρονα να εξαλείψουμε την παρουσία overfit ή underfit στο μοντέλο μας.
Έτσι, έπειτα από τους αντίστοιχους πειραματισμούς, ορίζουμε τα παρακάτω hyperparameters:

- EPOCHES = 20, επιλέξαμε λίγο μεγαλύτερο αριθμό epochs σε σχέση με τα μοντέλα του μέρους Α και του μέρους Β καθώς παρατηρήσαμε πως τόσο το train όσο και

το validation loss συνέχιζαν να μειώνονται κατα την πάροδο των epochs. Ωστόσο η μεγαλύτερη μείωση γίνεται κατα τα πρώτα epochs. Έπειτα από το 20ο epoch τόσο το train loss όσο και το validation loss παρέμεναν σταθερά (ή είχαν αμελητέα μείωση) για αυτό και σταματάμε την εκπαίδευση.

- BATCH_SIZE = 512, έπειτα από διάφορους πειραματισμούς καταλήξαμε σε αυτήν την τιμή με την οποία παρατηρήσαμε την βέλτιστη συμπεριφορά. Δοκιμάζοντας πιο μικρά μεγέθη (συνήθως δυνάμεις του 2) παρατηρήσαμε πως η εκπαίδευση καθυστερούσε αρκετά και τα αποτελέσματα δεν ήταν ικανοποιητικά, ενώ για μεγαλύτερα δεν παρατηρήθηκαν διαφορές στην απόδοση ωστόσο αυξανόταν ο αριθμός των epochs που χρειάζοντας έτσι ώστε το μοντέλο να έχει την επιθυμητή απόδοση.
- LEARNING_RATE = 0.0001, δοκιμάσαμε μεγαλύτερες τιμές, στις οποίες το μοντέλο λόγω της πιο γρήγορης εκμάθησης κατα την διάρκεια της εκπαίδευσης παρουσίαζε αστάθεια στα αποτελέσματα και πολλές φορές το μοντέλο παρουσίαζε overfit. Τα παραπάνω επιβεβαιώνονται παρακολουθώντας την πορεία των train και validation loss curves. Δοκιμάζοντας μικρότερες τιμές, το μοντέλο χρειαζόταν περισσότερα epochs για να εκπαιδευτεί και τελικά δεν είχε την επιθυμητή απόδοση.
- WINDOW SIZE = 10, η οποία παράμετρος αφορά ουσιαστικά τα time steps στα οποία θα χωριστούν οι χρονοσειρές, ώστε στη συνέχεια να φτιάξουμε τα κατάλληλα σύνολα (train και test). Δηλαδή το input του μοντέλου μας θα είναι κάθε φορά 10 συνεχόμενες τιμες μιας χρονοσειράς οι οποίες θα συμπιέζονται σε μια προκαθορισμένη διάσταση συμπίεσης (latent dimension). Δοκιμάσαμε τιμές από 10 έως 30. Καλύτερα αποτελέσματα παρατηρήσαμε χρησιμοποιώντας την τιμή 10.
  Περισσότερες λεπτομέρειες για το “σπάσιμο” της χρονοσειράς σε windows εξηγούνται στην συνέχεια.

Προ-επεξεργασία δεδομένων - Scaling

Στη συνέχεια, κατασκευάζουμε την δομή που θα χρησιμοποιηθεί για την εκπαίδευση του μοντέλου εφαρμόζοντας ταυτόχρονα και το απαραίτητο scaling (ώστε να κανονικοποιήσουμε τις τιμές των μετοχών).
<u>Παρατήρηση</u>: Η διαδικασία είναι αντίστοιχη με αυτήν του πρώτου και του δεύτερου μέρους, επομένως αρκετά πράγματα επαναλαμβάνονται.
Το μοντέλο μας είναι ένας auto-encoder, αυτό σημαίνει πως θέλουμε το τελικό output του μοντέλου να είναι όσο πιο κοντά στο αρχικό input. Δηλαδή να χάνεται όσο το δυνατόν μικρότερη πληροφορία κατα την διάρκεια της κωδικοποίησης και αποκωδικοποίησης.

Έτσι το y_train (μέσω του οποίου θα υπολογίζεται το loss του output του μοντέλου) θα είναι ο ίδιο με το input (δηλαδή με το X_train), έτσι ώστε τελικά να συγκρίνουμε το τελικό output με το αρχικό input. Στην πράξη δεν θα κατασκευάσουμε το y_train, αλλά αντί για (X_train,y_train) θα δίνουμε στην συνάρτηση fit το όρισμα (X_train,X_train) εφόσον τα δύο αυτά ταυτίζονται.
Όπως έχει αναφερθεί και παραπάνω το train set (και κατ' επέκταση τα X_train και y_train) κατασκευάζεται από το πρώτο 95% της κάθε χρονοσειράς. Έτσι στην παράγραφο αυτή, όταν αναφερόμαστε σε μία χρονοσειρά εννοούμε στο 95% των τιμών της χρονοσειράς.
Για την παραγωγή του X_train οι χρονοσειρές χωρίζονται σε διαδοχικά επικαλυπτώμενα sets μήκους ίσου με το WINDOW_SIZE (δηλαδή 10). Πιο συγκεκριμένα το πρώτο set τιμών αποτελείται από τις πρώτες window τιμές της χρονοσειράς, στο δεύτερο set έχει αφαιρεθεί η πρώτη τιμή του πρώτου set και έχει προστεθεί η window+1 τιμή της χρονοσειράς κ.ο.κ.. Το τελευταίο set περιέχει window τιμές οι οποίες αντιστοιχούν στις window τελευταίες τιμές της χρονοσειράς. Δηλαδή τα windows είναι επικαλυπτώμενα.
Έτσι, τελικά το X_train αποτελείται από sets όλων των χρονοσειρών.
Όπως είναι προφανές, κανένα set δεν αποτελείται από τιμές δύο διαφορετικών χρονοσειρών.

Για το scaling χρησιμοποιούμε τον MinMaxScaler, επιπλέον δοκιμάσαμε και τον StandardScaler, όμως η απόδοση του μοντέλου μας δεν ήταν τόσο ικανοποιητική (ωστόσο με μικρές διαφορές).
Όσον αφορά το scaling, εφαρμόζουμε ξεχωριστό scaling σε κάθε ένα window (συγκεκριμένα ανά 10 time steps). Δηλαδή ο scaler γίνεται fit σε κάθε ένα window ξεχωριστά. Το εύρος τιμών εσωτερικά ενός window είναι σχετικά μικρό. Έτσι οι scaled τιμές που προκύπτουν για κάθε ένα window είναι ομοιόμορφες σε κάθε περίπτωση. Πρόσθετα, πειραματιστήκαμε κάνοντας και scale ανά μετοχή, συγκρίνοντας τα αποτελέσματα του μοντέλου με το scale ανα window, στο οποίο τελικά καταλήξαμε αφού η απόδοση του μοντέλου ήταν αρκετά καλύτερη. Το scale ανά μετοχή δεν είναι τόσο αποδοτικό, διότι η αναπαράσταση που προκύπτει δεν αντιπροσωπεύει ακριβώς τις αρχικές τιμές της κάθε μετοχής, καθώς ορισμένες μετοχές έχουν μεγάλο εύρος τιμών ενώ άλλες έχουν πολύ μικρό με αποτέλεσμα οι scaled τιμές που προκύπτουν να μην είναι ομοιόμορφες μεταξύ των διαφορετικών χρονοσειρών.

Ορισμός του συνελικτικού νευρωνικού δικτύου αποκωδικοποίησης

Το συνελικτικό νευρωνικό δίκτυο περιλαμβάνει στρώματα συμπίεσης και αποσυμπίεσης.
Τόσο τα στρώματα συμπίεσης όσο και τα στρώματα αποσυμπίεσης αποτελούνται από convolutional layers μιας διάστασης (Conv1D). Οι βασικές παράμετροι για τον ορισμό ενός convolutional layer είναι:

- filters : η διάσταση του output που θα παραχθεί, που αντιστοιχεί στον αριθμό των filters στο convolution
- kernel_size : το μήκος του convolutional window

Κατά την συμπίεση, μέσω των convolutional layers αυξάνουμε το "πάχος" των δεδομένων μας ενώ κατα την αποσυμπίεση μειώνουμε το "πάχος".

Κατά την συμπίεση, ενδιάμεσα των convolutional layers υπάρχουν MaxPooling1D layers μέσω των οποίων πραγματοποιείται το DownSampling και έτσι επιτυγχάνεται η μείωση της πολυπλοκότητας/διάστασης του αρχικού window.

Αντίστοιχα κατά την αποσυμπίεση, ενδιάμεσα των convolutional layers υπάρχουν UpSampling1D layers μέσω των οποίων πραγματοποιείται το UpSampling και έτσι επιτυγχάνεται η αύξηση της πολυπλοκότητας/διάστασης του συμπιεσμένου window έτσι ώστε να το επαναφέρουμε στην αρχική του διάσταση.

Επίσης, έτσι ώστε να αυξήσουμε την απόδοση του μοντέλου προσθέσαμε:

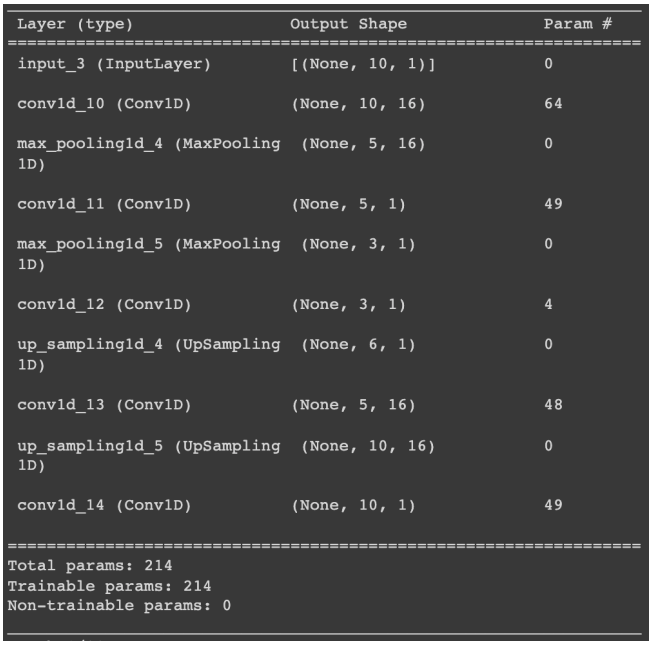
- ReLU activation function σε όλα τα convolutional layers (πλην του output convolutional layer - εξηγείται στην συνέχεια), καθώς παρατηρήσαμε βελτίωση στην απόδοση έπειτα από την προσθήκη τους.

Συνολικά, το συνελικτικό νευρωνικού δικτύου αποκωδικοποίησης αποτελείται από τα εξής:

- Input layer με shape αντίστοιχο του input, δηλαδή (10,1)
- Convolutional layer με filters=16, kernen_size=3 και relu activation function το οποίο αυξάνει το "πάχος" του input/window σε 16. Δηλαδή από το layer αυτό προκύπτει output με shape (10,16).
- Max pooling layer με pool_size=2 το οποίο μειώνει στο μισό την διάσταση/πολυπλοκότητα του input του. Δηλαδή από το layer αυτό προκύπτει output με shape (5,16).
- Convolutional layer με filters=1, kernen_size=3 και relu activation function το οποίο μειώνει το "πάχος" του input/window από 16 σε 1. Δηλαδή από το layer αυτό προκύπτει output με shape (5,1).

- Max pooling layer με pool_size=2 το οποίο μειώνει στο μισό την διάσταση/πολυπλοκότητα του input του. Λόγω της μη ακέραιας διαίρεσης της τρέχουσας διάστασης (5) με το 2, η διάσταση που προκύπτει είναι το άνω ακέραιο μέρος της διαιρεσης (δηλαδή 3). Έτσι το output που προκύπτει έχει shape (3,1).
  Έτσι έχουμε καταφέρει να δημιουργήσουμε το κωδικοποιημένο/συμπιεσμένο window το και να μειώσουμε την διάσταση του από 10 σε 3.
- Convolutional layer με filters=1, kernen_size=3 και relu activation function το οποίο αντιστοιχεί στο δεύτερο convolutional layer του encoder, κάνοντας την αντίστροφη διαδικασία. Η διάσταση παραμένει ίδια, δηλαδή (3,1).
- Upsampling layer με size=2 το οποίο διπλασιάζει την διάσταση/πολυπλοκότητα του input του. Δηλαδή το output που προκύπτει έχει shape (6,1).
- Convolutional layer με filters=16, kernen_size=2, relu activation function και padding=valid (δηλαδή χωρίς padding σε αντίθεση με τα υπόλοιπα Convolutional layers) έτσι ώστε να αυξήσουμε το “πάχος” των δεδομένων και ταυτόχρονα να μειώσουμε την διάσταση/πολυπλοκότητα από 6 σε 5. Δηλαδή το output που προκύπτει έχει shape (5,16).
- Upsampling layer με size=2 το οποίο διπλασιάζει την διάσταση/πολυπλοκότητα του input του. Δηλαδή το output που προκύπτει έχει shape (10,16).
- Convolutional layer με filters=1, kernen_size=3 και sigmoid activation function έτσι ώστε να μειώσουμε το “πάχος” των δεδομένων και τελικά να έρθει στην “μορφή” του αρχικού input, δηλαδή να προκύψει το τελικό αποκωδικοποιημένο window. Επομένως το output που προκύπτει έχει shape (10,1).

Αναλυτικότερα η εικόνα του μοντέλου, όπως παρουσιάζεται από την συνάρτηση model.summary(), είναι η εξής:



Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 10, 1)]	0
conv1d_10 (Conv1D)	(None, 10, 16)	64
max_pooling1d_4 (MaxPooling 1D)	(None, 5, 16)	0
conv1d_11 (Conv1D)	(None, 5, 1)	49
max_pooling1d_5 (MaxPooling 1D)	(None, 3, 1)	0
conv1d_12 (Conv1D)	(None, 3, 1)	4
up_sampling1d_4 (UpSampling 1D)	(None, 6, 1)	0
conv1d_13 (Conv1D)	(None, 5, 16)	48
up_sampling1d_5 (UpSampling 1D)	(None, 10, 16)	0
conv1d_14 (Conv1D)	(None, 10, 1)	49

Total params: 214
Trainable params: 214
Non-trainable params: 0

Εκπαίδευση του μοντέλου

Όσον αφορά την εκπαιδευση του μοντέλου αρχικά ορίζουμε τον optimizer καθώς και το loss function που θα χρησιμοποιηθούν. Πιο συγκεκριμένα, ορίζουμε:

- Optimizer : Adam , δίνοντας του σαν παράμετρο το learning rate το οποίο ορίσαμε παραπάνω.
- Loss Function : Binary Cross Entropy loss

Επίσης κατά την κλήση της συνάρτησης fit (η οποία πραγματοποιεί την εκπαίδευση), ορίζουμε τις εξής παραμέτρους:

- Το x_train δίνεται τόσο στην παράμετρο x όσο και την παράμετρο y. Εξηγείται αναλυτικά στη παράγραφο Προ-επεξεργασία δεδομένων - Scaling.
- epochs και batch_size τα οποία έχουν οριστεί στον Ορισμός υπερ-παραμέτρων.
- validation_data=(x_test,x_test) , δηλαδή χρησιμοποιούμε το test set σαν validation set έτσι ώστε να παρακολουθούμε την πορεία του μοντέλου κατα την εκπαιδευση του πάνω σε άγνωστα δεδομένα. Έτσι, καταφέρνουμε να γενικεύσουμε το μοντέλο μας, δηλαδή να μην παρουσιάζει overfit ή underfit. Ο ορισμός του x_test εξηγείται στην παράγραφο Προ-επεξεργασία δεδομένων - Scaling.

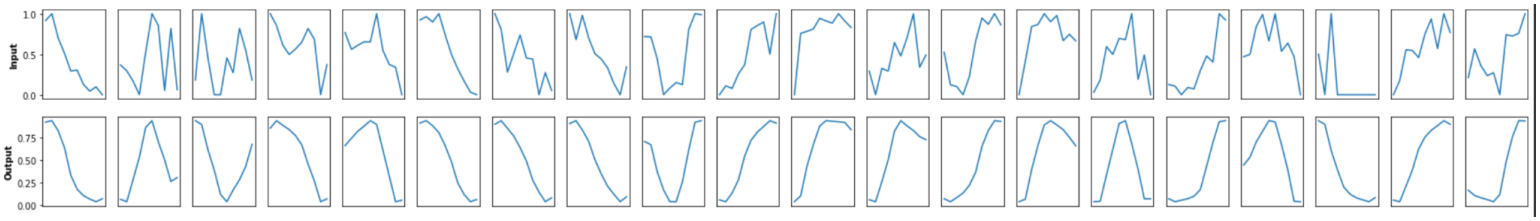
Εκπαιδευμένο μοντέλο

Με βάση τα παραπάνω έχουμε εκπαιδεύσει ένα μοντέλο χρησιμοποιώντας όλες τις χρονοσειρές του dataset που μας δόθηκαν.
Για την συμπίεση των χρονοσειρών χρησιμοποιούμε μόνο τον encoder του μοντέλου. Δηλαδή, δίνοντας ένα window στον encoder, το output αντιστοιχεί στο συμπιεσμένο window. Συνενώνοντας τα window αυτά, προκύπτει τελικά η συμπιεσμένη χρονοσειρά. Περισσότερες λεπτομέρειες εξηγούνται στην συνέχεια.
Ο decoder χρησιμοποιείται μόνο κατά την εκπαίδευση, έτσι ώστε η κωδικοποιημένη αναπαράσταση που προκύπτει από τον encoder να περιέχει όσο το δυνατόν περισσότερη πληροφορία από το αρχικό input.
Έτσι, αποθηκεύουμε ξεχωριστά το μοντέλο του encoder έπειτα από την εκπαιδευση. Το μοντέλο του encoder το έχουμε αποθηκεύσει στο αντίστοιχο αρχείο models/part3/part3_encoder.h5 και χρησιμοποιείται από το εκτελέσιμο αρχείο του τρίτου μέρους της εργασίας (όπως εξηγείται και στην συνέχεια).
Στον φάκελο αυτό υπάρχει αποθηκευμένο και ολόκληρο το μοντέλο του autoencoder, το οποίο όμως δεν χρησιμοποιείται από το εκτελέσιμο πρόγραμμα.

Αξιολόγηση του μοντέλου

Όπως αναφέρθηκε και παραπάνω η εκπαίδευση το μοντέλου γίνεται χρησιμοποιώντας το πρώτο 95% του κάθε curve.
Έπειτα από την εκπαιδευση του μοντέλου, χρησιμοποιούμε το test set (δηλαδή το υπόλοιπο 5% της χρονοσειράς) για να ελέγξουμε την απόδοση του συνολικού μοντέλου του autoencoder.
Πιο συγκεκριμένα δίνοντας στο μοντέλο κάποια τυχαία windows (δηλαδή 10αδες τιμών) του test set, αυτό μας επιστρέφει τα αντίστοιχα αποκωδικοποιημένα windows.
Έτσι συγκρίνουμε το output του μοντέλου σε σχέση με το onput που του δώσαμε.
Στην συνέχεια παρουσιάζω κάποια από αυτά τα παραδείγματα.
Στην εικόνα που ακολουθεί:

- Κάθε στήλη αντιστοιχεί σε ένα τυχαίο window. Η πάνω εικόνα αντιστοιχεί στο window εισόδου ενώ η κάτω εικόνα αντιστοιχεί στο αποκωδικοποιημένο window που προέκυψε από το μοντέλο.
- Οι τιμές που παρουσιάζονται στα γραφήματα τόσο του input window όσο και του output window είναι σε scaled μορφή στο διάστημα (0,1) όπως έχει αναφερθεί και στην Προ-επεξεργασία δεδομένων - Scaling.



Παρατηρούμε ότι στα αποκωδικοποιημένα windows η βασική πληροφορία όσον αφορά την πορεία του curve συνεχίζει να υπάρχει. Κάποιες ενδιάμεσες ανωμαλίες απουσιάζουν από το output window, ωστόσο αυτές δεν αποτελούν σημαντική πληροφορία.
Γενικότερα παρατηρούμε πολύ καλή απόδοση καθώς τα windows του output ακολουθούν σχεδόν σε όλες τις περιπτωσεις παρόμοια πορεία.
Αυτό σημαίνει ότι κατα την διάρκεια της κωδικοποίησης - αποκωδικοποίησης χάνεται πολύ λίγη πληροφορία.

Παρουσίαση συμπιεσμένων χρονοσειρών

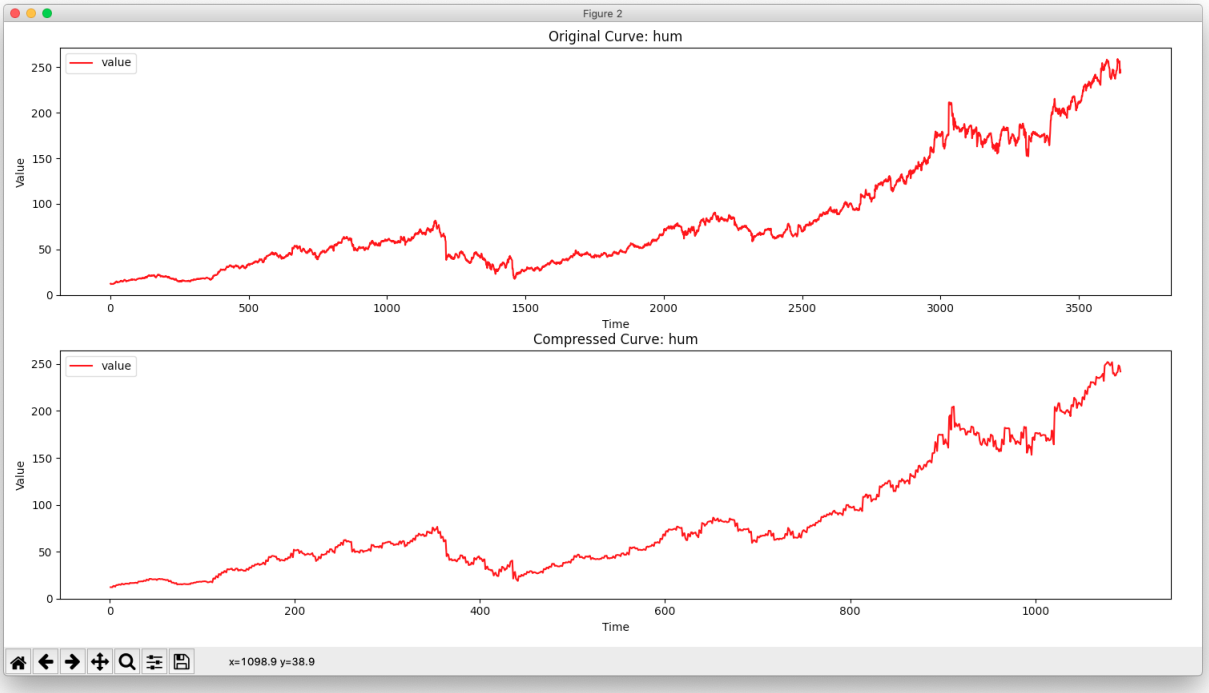
Στην συνέχεια θα παρουσιάσουμε κάποιες συμπιεσμένες χρονοσειρές.
Η διαδικασία που ακολουθείται για την δημιουργία μιας συμπιεσμένης χρονοσειράς είναι η εξής:

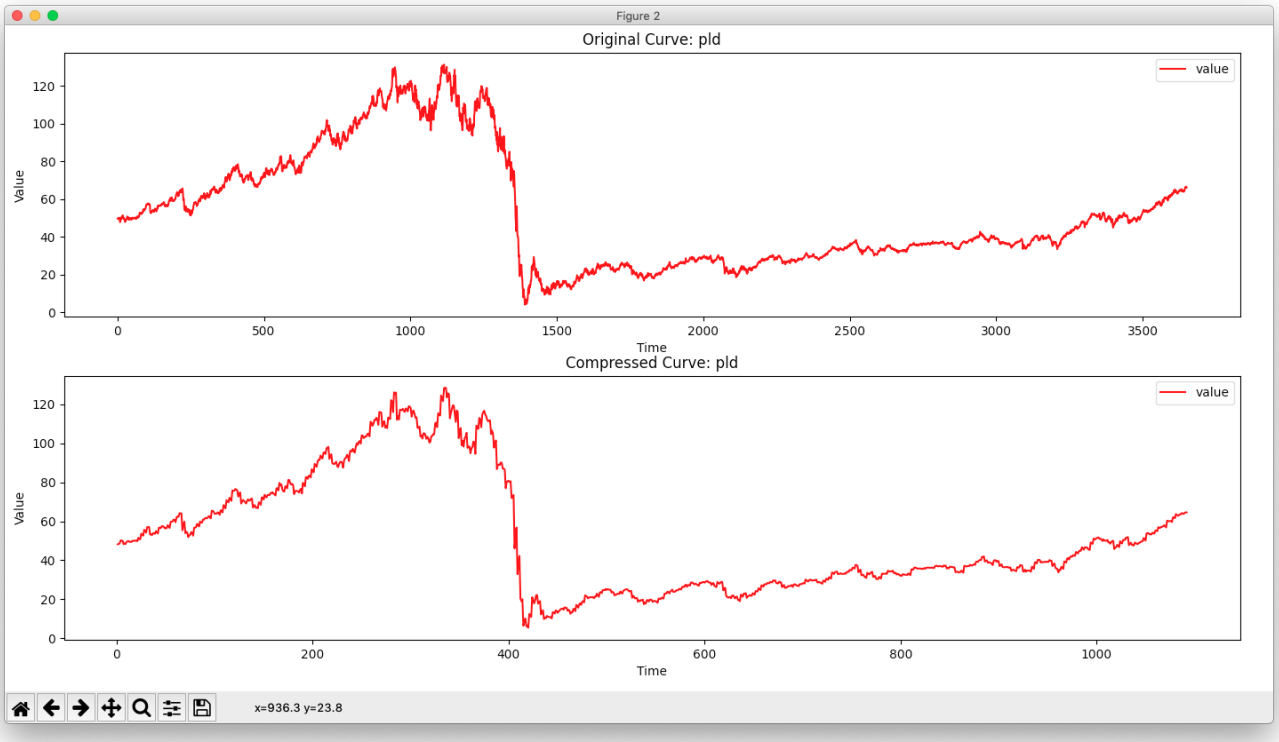
- Χρησιμοποιώντας ολόκληρη την χρονοσειρά την χωρίζουμε σε ΜΗ επικαλυπτόμενα windows. Έχουμε ορίσει το window size ίσο με 10, επομένως το πρώτο window αντιστοιχεί στις πρώτες 10 τιμές της χρονοσειράς, το δεύτερο window στις επόμενες 10 τιμές [10,20] κ.ο.κ.. Στην ουσία “διαιρούμε” την χρονοσειρά σε 10αδες.

- Εφαρμόζουμε scaling σε κάθε ένα window ξεχωριστά, δηλαδή ο scaler γίνεται fit σε κάθε window. Για κάθε ένα window αποθηκεύουμε τον αντίστοιχο scaler που χρησιμοποιήθηκε και έγινε fit “πάνω” σε αυτό, έτσι ώστε να τον χρησιμοποιήσουμε στην συνέχεια για το unscaling.
- Εκτελούμε την συνάρτηση predict χρησιμοποιώντας τον encoder (και όχι ολόκληρο το μοντέλο του autoencoder) και δίνοντας σαν είσοδο τα windows που προέκυψαν από την χρονοσειρά. Έτσι για κάθε ένα window παίρνουμε σαν output το αντίστοιχο συμπιεσμένο window (μεγέθους 3).
- Κάνουμε unscale τα windows που μας επιστράφηκαν από τον encoder. Για κάθε ένα window χρησιμοποιούμε τον αντίστοιχο scaler που χρησιμοποιήθηκε (και έγινε fit) κατά το scaling και αποθηκεύτηκε στο παραπάνω βήμα.
- Τέλος, ενώνοντας (δηλαδή concatenate) τα windows προκύπτει η συμπιεσμένη χρονοσειρά.

Όσον αφορά το dataset που μας δόθηκε, οι χρονοσειρές έχουν μήκος 3650. Επομένως κατα τον χωρισμό της χρονοσειράς σε windows προκύπτουν 365 windows (μηκους 10 το κάθε ένα). Εφόσον το κάθε window συμπιέζεται από διάσταση 10 σε διάσταση 3, το μήκος της τελικής συμπιεσμένης χρονοσειράς είναι 365*3 = 1095.

Στην συνέχεια παρουσιάζουμε τις συμπιεσμένες χρονοσειρες για δύο τυχαίες χρονοσειρές του dataset. Κάθε φορά που παρουσιάζουμε μια συμπιεσμένη χρονοσειρά, αρχικά (στο πάνω γράφημα) παραθέτουμε την αρχική χρονοσειρά (μηκους 3650) και έπειτα (στο κάτω γράφημα) παραθέτουμε την συμπιεσμένη χρονοσειρά.





Και στις δύο περιπτώσεις παρατηρούμε πως η συμπιεσμένη χρονοσειρά ακολουθεί ακριβώς την ίδια πορεία με την αρχική χρονοσειρά, με ελάχιστες ίσως διαφορές. Δηλαδή παρατηρούμε πως η χρονοσειρά εξακολουθεί να έχει όλη την πληροφορία από την αρχική χρονοσειρά όσον αφορά τις αυξομειώσεις των τιμών ενώ ταυτόχρονα έχει μειωθεί η διάσταση της από 3650 σε 1095.
Έτσι επιβεβαιώνεται η πολύ καλή απόδοση του μοντέλου μας.

Δημιουργία των output αρχείων csv

Χρησιμοποιώντας τα αρχεία dataset και query που δόθηκαν σαν είσοδος κατα την εκτέλεση του προγράμματος δημιουργούμε τα αντίστοιχα αρχεία τα οποία περιέχουν τις ίδιες χρονοσειρές αλλά στην συμπιεσμένη τους “μορφή”.
Πιο συγκεκριμένα, για την δημιουργία του output dataset αρχείου:

- Για κάθε μια χρονοσειρά του αρχείου dataset που δόθηκε, ακολουθείται η διαδικασία που περιγράφεται στην παράγραφο Παρουσίαση συμπιεσμένων χρονοσειρών έτσι ώστε να παραχθεί η αντίστοιχη συμπιεσμένη χρονοσειρά.
- Έχοντας εκτελέσει το παραπάνω για κάθε μια χρονοσειρά του dataset, έχουμε όλες τις αντιστοιχες συμπιεσμενες χρονοσειρές. Έτσι, κατασκευάζουμε ένα νέο dataframe το οποίο τις περιέχει.
- Μέσω του dataframe (και της συνάρτησης to_csv) δημιουργούμε το αντίστοιχο csv αρχείο output_dataset.

Την ίδια διαδικασία ακολουθούμε και για το query dataset που μας δόθηκε έτσι ώστε να παραχθεί το csv αρχείο output_query.

Παραδοτέο αρχείο - reduce.py

Η διαδικασία που ακολουθείται κατα την εκτέλεση του παραδοτέου αρχείου είναι:

- Αρχικά διαβάζονται οι παράμετροι που έδωσε ο χρήστης από την γραμμή εντολων. Οι παράμετροι είναι:
    - -d <dataset> : και αντιστοιχεί στο path του dataset το οποίο θα χρησιμοποιηθεί για την Δημιουργία των output αρχείων csv.
    - -q <queryset> : και αντιστοιχεί στο path του dataset το οποίο θα χρησιμοποιηθεί για την Δημιουργία των output αρχείων csv.
    - -od <output_dataset_file> : και αντιστοιχεί στο path του output dataset file το οποίο θα δημιουργηθεί.
    - -oq <output_query_file> : και αντιστοιχεί στο path του output query file το οποίο θα δημιουργηθεί.
    - -n <number of time series to be presented> : επιπλέον προαιρετική παράμετρος που προσθέσαμε έτσι ώστε να παρουσιάζονται γραφικά κάποιες συμπιεσμένες χρονοσειρές. Εξηγείται αναλυτικότερα στην συνέχεια.
- Φορτώνεται το μοντέλο του auto encoder οποίο έχει ήδη εκπαιδευτεί. *Παρατήρηση: αλλάζοντας την τιμή της μεταβλητής TRAIN_NEW_MODEL σε True, αντί να φορτωθεί το εκπαιδευμένο μοντέλο εκπαιδεύεται ένα νέο μοντέλο με βάση το dataset που έχει δοθεί χρησιμοποιώντας τις χρονοσειρές του dataset αρχείου. Ο κώδικας που εκτελείται στην περίπτωση αυτή είναι αυτός που χρησιμοποιήσαμε για την εκπαιδευση του μοντέλου μας. Η τιμή της μεταβλητής TRAIN_NEW_MODEL είναι ορισμένη σε False, έτσι ώστε να φορτώνεται το ήδη εκπαιδευμένο μοντέλο κατά την εκτέλεση του προγράμματος.*
- Στην συνέχεια εκτελείται η διαδικασία που περιγράφεται στην αντίστοιχη παράγραφο για την Δημιουργία των output αρχείων csv. Αρχικά δημιουργείται το output_query_file και την συνέχεια το output_dataset_file.
- Τέλος, παρουσιάζονται γραφικά κάποιες χρονοσειρές έτσι ώστε να επιδείξουμε την απόδοση του μοντέλου μας. Ανάλογα με την τιμή που δόθηκε στην προαιρετική παράμετρο **-n** την οποία προσθέσαμε, τόσες είναι και οι χρονοσειρές που θα παρουσιαστούν. Αν δεν δοθεί η παράμετρος -n έχουμε ορίσει default τιμή n=4.
  Πιο συγκεκριμένα, επιλέγουμε **n** τυχαίες χρονοσειρές απο το dataset αρχείο που μας δοθηκε και για κάθε μία από αυτές εκτελούμε την διαδικασία που περιγράφεται αναλυτικά στην παράγραφο Παρουσίαση συμπιεσμένων χρονοσειρών. Δηλαδή παρουσιάζουμε την αρχική καθώς και την συμπιεσμένη χρονοσειρά έτσι ώστε να τις συγκρίνουμε.

Μέρος Δ - Comparison of two representations

Εκτελώντας το παραδοτέο αρχείο του Μέρους Γ και δίνοντας του σαν input:

- dataset file : τις πρώτες 339 χρονοσειρές του αρχείου nasdaq2007_17.csv που μας δόθηκε
- dataset file : τις τελευταίες 10 χρονοσειρές του αρχείου nasdaq2007_17.csv που μας δόθηκε

προκύπτουν τα αντίστοιχα output dataset και output query file με τις αντίστοιχες συμπιεσμένες χρονοσειρές.
Χρησιμοποιώντας τόσο τα αρχικά dataset/query files όσο και τα output dataset/query files και εκτελώντας όλα τα ερωτήματα του παραδοτέου της 2ης εργασίας, προκύπτουν 19 αρχεία output (2 για κάθε ερώτημα), τα οποία περιλαμβάνονται στο φάκελο με όνομα “part4”, προκειμένου τα συγκρίνουμε την νέα αναπαράσταση για κάθε μετοχή / χρονοσειρά, που προέκυψε από τον encoder του ερωτήματος 3, με την αρχική αναπαρασταση. Όπως αναφέρεται και παραπάνω η νέα αναπαράσταση των μετοχών / χρονοσειρών που προκύπτει έχει διάσταση ίση με 1095, ενώ η αρχική έχει διάσταση ίση με 3650. Συνεπώς, λόγω της μείωσης αυτής παρατηρήσαμε κυρίως αρκετά μεγάλη διαφορά στους χρόνους μεταξύ των εκτελέσεων αρχικής και νεας αναπαράστασης, ωστόσο παρά την μείωση των διαστάσεων δεν αλλοιώνονται οι μετοχές / χρονοσειρές και αυτό γίνεται αντιληπτό από τα αποτελέσματα που παρουσιάζουμε στην στην συνέχεια τόσο στην αναζήτηση πλησιέστερου γείτονα όσο και στην συσταδοποίηση. Ακολουθούν, λοιπόν, συγκριτικά τα αποτελέσματα από τις εκτελέσεις για τις 2 αυτές αναπαραστάσεις.

Αναζήτηση πλησιέστερου γείτονα

Vectors - LSH με χρήση της μετρικής L2

Όνομα Μετοχής / Χρονοσειράς	Approximate Nearest neighbor με την αναπαράσταση με 3650d	Approximate Distance	True Distance	approximate/true distance	Approximate Nearest neighbor με την αναπαράσταση με 1095d	Approximate Distance	True Distance	approximate/true distance	True Nearest neighbor με 3650d/ 1095d
x	ncr	1943.494794	948.083083	2.049	ceco	1156.289535	508.201510	2.275	rig/rig
xel	cce	214.303045	210.812841	1.016	cce	117.771682	113.902907	1.033	t/t
xl	adm	1523.018049	560.961987	2.715	fitb	568.272824	300.185506	1.893	sti/sti
xlnx	pki	331.205943	306.839476	1.079	snps	166.470970	166.470970	1.0	snps/snps
xom	hsic	1654.617272	586.204632	2.822	axp	559.671548	315.598228	1.773	slb/slb
xray	msa	476.102457	274.592610	1.733	aep	154.115096	146.488566	1.052	pcg/pcg
xrx	pfg	756.264209	627.549170	1.205	pfg	407.798466	338.931935	1.203	kss/kss
yum	emn	710.571512	402.687561	1.7645	dri	228.175629	216.254398	1.055	gis/gis
zbh	antm	1342.020735	669.378698	2.004	pep	531.487382	354.393539	1.499	ups/ups
zion	lamr	1292.784710	445.542967	2.901	txt	613.397435	242.262461	2.531	sti/sti

Για την Αναπαράσταση με 3650d έχουμε τα εξής:

tApproximateAverage: 0.000149 seconds
tTrueAverage: 0.001650 seconds
Max Approximation Factor: 2.901594
Min Approximation Factor: 1.016556

Για την Αναπαράσταση με 1095d έχουμε τα εξής:

tApproximateAverage: 0.000055 seconds
tTrueAverage: 0.000459 seconds
Max Approximation Factor: 2.531954
Min Approximation Factor: 1.000000

Παρατηρήσεις

- Συγκρίνοντας τα tApproximateAverage και το tTrueAverage των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/5 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Έπειτα, παρατηρώντας τον παραπάνω πίνακα, και κυρίως συγκρίνοντας τις 2 στήλες (όπου η κάθε μια αφορά την εκτέλεση για την αντίστοιχη αναπαράσταση) που έχουμε υπολογίσει τον λόγο approximate/true distance (Approximation Factor), συμπεραίνουμε ουσιαστικά πως παρά την μείωση των διαστάσεων από 3650 σε 1095 δεν χάθηκε πληροφορία κατά το encoding, καθώς τα approximation factors της εκτέλεσης με την νέα αναπαράστασης έχουν ελάχιστη διαφορά σε σχέση με τα αντίστοιχα της εκτέλεσης με την αρχική αναπαράσταση, μάλιστα μερικά είναι βελτιωμένα προς το καλύτερο (δηλαδή είναι “μικρότερα” από τα αρχικά). Τέλος, παρατηρούμε πως για την μετοχή xlnx, ενώ το πρόγραμμα με χρήση της αρχικής αναπαράστασης κάνει μια πολύ καλή προσέγγιση βρίσκοντας έναν appoximate neighbor (pki) κοντά στον αληθινό (Approximation Factor: 1.079), με χρήση της νέας αναπαράστασης βρίσκει τον αληθινό ως appoximate neighbor (είναι υπογραμμισμένο στον πίνακα).
- Επίσης, βλέποντας την τελευταία στήλη του πίνακα ”True Nearest neighbor για 3650d/1095d” είναι φανερό πως οι εκτελέσεις του προγράμματος και με τις δύο αναπαραστάσεις βρίσκουν τους ίδιους true nearest neighbor, αυτό σημαίνει πως στην νέα αναπαράσταση παρά την μείωση των διαστάσεων οι μετοχές δεν έχασαν της πληροφορία τους κατά το encoding, έτσι που μέσω της Brute Force Μεθόδου με χρήση της μετρικής L2 το πρόγραμμά μας καταφέρνει να βρει του ίδιους κοντινότερους γείτονες σε σχέση με την εκτέλεση για την αρχική αναπαράσταση.

Vectors - Hypercube με χρήση της μετρικής L2

Όνομα Μετοχής / Χρονοσειράς	Approximate Nearest neighbor με την αναπαράσταση με 3650d	Approximate Distance	True Distance	approximate/true distance	Approximate Nearest neighbor με την αναπαράσταση με 1095d	Approximate Distance	True Distance	approximate/true distance	True Nearest neighbor με 3650d/ 1095d
x	pld	1566.599528	948.083083	1.652	nav	740.035004	508.201510	1.4561	rig/rig
xel	cce	214.303045	210.812841	1.016	cce	117.771682	113.902907	1.033	t/t
xl	aa	871.888534	560.961987	1.554	hig	430.515539	300.185506	1.4341	sti/sti
xlnx	snps	306.839476	306.839476	1.0	payx	192.597137	166.470970	1.1569	snps/snps
xom	slb	586.204632	586.204632	1.0	cvx	443.304492	315.598228	1.4046	slb/slb
xray	aep	288.075738	274.592610	1.049	aep	154.115096	146.488566	1.052	pcg/pcg
xrx	hig	881.575244	627.549170	1.404	kss	338.931935	338.931935	1.0	kss/kss
yum	gis	402.687561	402.687561	1.0	dri	228.175629	216.254398	1.055	gis/gis
zbh	swk	1253.371127	669.378698	1.8724	hsy	674.418212	354.393539	1.903	ups/ups
zion	aa	744.377058	445.542967	1.670	sti	242.262461	242.262461	1.0	sti/sti

Για την Αναπαράσταση με 3650d έχουμε τα εξής:

tApproximateAverage: 0.000248 seconds
tTrueAverage: 0.001714 seconds
Max Approximation Factor: 1.872440
Min Approximation Factor: 1.000000

Για την Αναπαράσταση με 1095d έχουμε τα εξής:

tApproximateAverage: 0.000070 seconds
tTrueAverage: 0.000465 seconds
Max Approximation Factor: 1.903021
Min Approximation Factor: 1.000000

Παρατηρήσεις

- Συγκρίνοντας τα tApproximateAverage και το tTrueAverage των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/4 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Έπειτα, παρατηρώντας τον παραπάνω πίνακα, και κυρίως συγκρίνοντας τις 2 στήλες (όπου η κάθε μια αφορά την εκτέλεση για την αντίστοιχη αναπαράσταση) που έχουμε υπολογίσει τον λόγο approximate/true distance (Approximation Factor), συμπεραίνουμε ουσιαστικά πως παρά την μείωση των διαστάσεων από 3650 σε 1095 δεν χάθηκε πληροφορία κατά το encoding, καθώς τα approximation factors της εκτέλεσης με την νέα αναπαράστασης έχουν ελάχιστη διαφορά σε σχέση με τα αντίστοιχα της εκτέλεσης με την αρχική αναπαράσταση, μάλιστα μερικά είναι βελτιωμένα προς το καλύτερο (δηλαδή είναι “μικρότερα” από τα αρχικά). Αναλυτικότερα, για τις χρονοσειρές xlnx, yum και xom η εκτέλεση με χρήση της νεα αναπαράσταση δεν βρίσκει ως approximate neighbors τους true neighbors, σε αντίθεση με την εκτέλεση με χρήση της αρχικής, η οποία όπως βλέπουμε τους βρίσκει με επιτυχία. Βέβαια, με την νέα αναπαράσταση γίνεται μια αρκετά καλή προσέγγιση των approximate neighbors αυτών των 2 χρονοσειρών καθώς οι approximation factors είναι οι εξής 1.1569, 1.055 και 1.4046. Τέλος, σε αντίθεση με τα αποτελέσματα για την αναπαρασταση με τα 3650d, με την νέα αναπαράσταση μειωμένων διαστάσεων το πρόγραμμα βρίσκει για τις χρονοσειρές xrx και zion ως approximate neighbors τους true neighbors.
- Επίσης, βλέποντας την τελευταία στήλη του πίνακα ”True Nearest neighbor για 3650d/1095d” είναι φανερό πως οι εκτελέσεις του προγράμματος και με τις δύο αναπαραστάσεις βρίσκουν τους ίδιους true nearest neighbor, αυτό σημαίνει πως στην νέα αναπαράσταση παρά την μείωση των διαστάσεων οι μετοχές δεν έχασαν της πληροφορία τους κατά το encoding, έτσι που μέσω της Brute Force Μεθόδου με χρήση της μετρικής L2 το πρόγραμμά μας καταφέρνει να βρει του ίδιους κοντινότερους γείτονες σε σχέση με την εκτέλεση για την αρχική αναπαράσταση.

Time Series - LSH με χρήση της μετρικής Discrete Frechet

Όνομα Μετοχής / Χρονοσειράς	Approximate Nearest neighbor με την αναπαράσταση με 3650d	Approximate Distance	True Distance	approximate/true distance	Approximate Nearest neighbor με την αναπαράσταση με 1095d	Approximate Distance	True Distance	approximate/true distance	True Nearest neighbor με 3650d/ 1095d
x	bb	52.694555	52.694555	1.0	epc	94.233462	43.600409	2.1612	bb/bb
xel	cce	11.553359	7.295700	1.5835	cms	7.154802	7.154802	1.0	cms/cms
xl	mtg	35.852827	34.366000	1.0432	snv	44.824694	28.292541	1.5843	sti/sti
xlnx	snps	16.459406	14.166000	1.1618	a	13.124230	13.124230	1.0	a/a
xom	oxy	25.080130	24.085000	1.0413	pg	20.061908	20.061908	1.0	pg/pg
xray	pcg	15.857628	15.857628	1.0	payx	13.177261	13.177261	1.0	pcg/payx
xrx	mdp	35.921492	22.797194	1.5756	fe	26.728014	21.198409	1.2608	kss/kss
yum	vfc	20.721000	16.660328	1.2437	dri	16.271400	15.917106	1.0222	tmk/tmk
zbh	etr	56.151482	28.055000	2.0014	ir	58.226597	26.919862	2.1629	ups/ups
zion	sti	16.953000	16.953000	1.0	sti	16.091160	16.091160	1.0	sti/sti

Για την Αναπαράσταση με 3650d έχουμε τα εξής:

tApproximateAverage: 3.394898 seconds
tTrueAverage: 33.046057 seconds
Max Approximation Factor: 2.001479
Min Approximation Factor: 1.000000

Για την Αναπαράσταση με 1095d έχουμε τα εξής:

tApproximateAverage: 0.656459 seconds
tTrueAverage: 2.956888 seconds
Max Approximation Factor: 2.162960
Min Approximation Factor: 1.000000

Παρατηρήσεις

- Συγκρίνοντας τα tApproximateAverage και το tTrueAverage των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/5 για το tApproximateAverage και στο 1/11 για το tTrueAverage σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Έπειτα, παρατηρώντας τον παραπάνω πίνακα, και κυρίως συγκρίνοντας τις 2 στήλες (όπου η κάθε μια αφορά την εκτέλεση για την αντίστοιχη αναπαράσταση) που έχουμε υπολογίσει τον λόγο approximate/true distance (Approximation Factor), συμπεραίνουμε ουσιαστικά πως παρά την μείωση των διαστάσεων από 3650 σε 1095 δεν χάθηκε πληροφορία κατά το encoding, καθώς τα approximation factors της εκτέλεσης με την νέα αναπαράστασης έχουν ελάχιστη διαφορά σε σχέση με τα αντίστοιχα της εκτέλεσης με την αρχική αναπαράσταση, μάλιστα μερικά είναι βελτιωμένα προς το καλύτερο (δηλαδή είναι “μικρότερα” από τα αρχικά). Αναλυτικότερα, η εκτέλεση με χρήση της νεας αναπαράσταση που βρίσκει επιτυχημένα για 5 χρονοσειρές (xel, xlnx, xom, xray, zion) ως approximate neighbors τους true neighbors, σε αντίθεση με την εκτέλεση με χρήση της αρχικής, η οποία όπως βλέπουμε βρίσκει με επιτυχία 3 χρονοσειρές (x, xray, zion). Συμπεραίνοντας πως η νέα αναπαράσταση των μετοχών είναι αισθητά πιο βελτιωμένη από την αρχική.
- Επίσης, βλέποντας την τελευταία στήλη του πίνακα ”True Nearest neighbor για 3650d/1095d” είναι φανερό πως οι εκτελέσεις του προγράμματος και με τις δύο αναπαραστάσεις βρίσκουν τους ίδιους true nearest neighbor, αυτό σημαίνει πως στην νέα αναπαράσταση παρά την μείωση των διαστάσεων οι μετοχές δεν έχασαν της πληροφορία τους κατά το encoding, έτσι που μέσω της Brute Force Μεθόδου με χρήση της μετρικής Discrete Frechet το πρόγραμμά μας καταφέρνει να βρει σχεδον τους ίδιους κοντινότερους γείτονες σε σχέση με την εκτέλεση για την αρχική αναπαράσταση.

Time Series - LSH με χρήση της μετρικής Continuous Frechet

Όνομα Μετοχής / Χρονοσειράς	Approximate Nearest neighbor με την αναπαράσταση με 3650d	Approximate Distance	True Distance	approximate/true distance	Approximate Nearest neighbor με την αναπαράσταση με 1095d	Approximate Distance	True Distance	approximate/true distance	True Nearest neighbor με 3650d/ 1095d
x	lvlt	77.136000			gild	56.840178			
xel	cms	6.132900			bll	8.959884			
xl	sti	20.366000			bac	38.440850			
xlnx	bax	13.646000			expd	12.143358			
xom	chrw	13.288621			chrw	13.408332			
xray	pcar	9.091000			wfc	11.274381			
xrx	exc	15.187000			jhg	15.245648			
yum	tmk	8.011000			msft	10.008827			
zbh	adp	27.372734			pep	21.737761			
zion	sti	12.830000			teva	32.531842			

Για την Αναπαράσταση με 3650d έχουμε τα εξής:

tApproximateAverage: 650.603310 seconds

Για την Αναπαράσταση με 1095d έχουμε τα εξής:

tApproximateAverage: 22.347464 seconds

Παρατηρήσεις

- Συγκρίνοντας τα tApproximateAverage των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/30 για το tApproximateAverage σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Ο υπολογισμός των αποστάσεων μέσω της Brute Force μεθόδου με χρήση της μετρικής Continuous Frechet είναι πάρα πολύ χρονοβόρα με αποτέλεσμα να μην μπορέσουμε να κάνουμε τις ανάλογες μετρήσεις.

Συσταδοποίηση Μετοχών / Χρονοσειρών

Vectors - Lloyd's - Υπολογισμός της μέσης χρονοσειράς ως διάνυσμα

Εκτέλεση αναπαράστασης με 3650d				**Εκτέλεση αναπαράστασης με 1096d**			
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 1	Cluster 2	Cluster 3	Cluster 4
346	1	1	1	345	1	2	1
Clustering time: 0.031184 seconds				**Clustering time:** 0.014093 seconds			
Silhouette				**Silhouette**			
0.972528	1.00	1.00	1.00	0.980899	1.00	0.430666	1.00
stotal = 0.972764				stotal = 0.977855			

Παρατηρήσεις

- Συγκρίνοντας τα Cluster time των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/3 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Παράλληλα συγκρίνοντας τις σιλουέτες (stotal) των δύο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, παρατηρούμε ότι και στις 2 περιπτώσεις έχουμε τέλεια σιλουέτα που σημαίνει πως έγινε πολύ καλό clustering. Συμπεραίνουμε, λοιπόν, πως δεν υπάρχουν άλλα περιθώρια βελτίωσης, δηλαδή ότι το clustering δεν μπορεί να βελτιωθεί χρησιμοποιώντας την αναπαράσταση με τις μειωμένες διαστάσεις. Παρόλα αυτά, όπως αναφέραμε παραπάνω, καταφέραμε να μειώσουμε αισθητά τον χρόνο μέσω της νέας αναπαράστασης και μάλιστα χωρίς να προκύψουν αλλοιώσεις μεταξύ των μετοχών της αρχικής και της νέας, αν συνέβαινε κάτι τέτοιο ίσως η εκτέλεση του clustering για την αναπαράσταση με τις μειωμένες διαστάσεις μας οδηγούσε σε διαφορετική συσταδοποίηση των μετοχών.

Time Series - Lloyd's - Υπολογισμός της μέσης χρονοσειράς ως καμπύλη

Εκτέλεση αναπαράστασης με 3650d				**Εκτέλεση αναπαράστασης με 1096d**			
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 1	Cluster 2	Cluster 3	Cluster 4
345	1	1	2	345	1	1	2
Clustering time: 692.024080 seconds				**Clustering time:** 57.494841 seconds			
Silhouette				**Silhouette**			
0.976173	1.00	1.00	0.599784	0.976233	1.00	1.00	0.598363
stotal = 0.974152				stotal = 0.977855			

Παρατηρήσεις

- Συγκρίνοντας τα Cluster time των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/12 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Παράλληλα συγκρίνοντας τις σιλουέτες (stotal) των δύο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, παρατηρούμε ότι και στις 2 περιπτώσεις έχουμε τέλεια σιλουέτα που σημαίνει πως έγινε πολύ καλό clustering. Συμπεραίνουμε, λοιπόν, πως δεν υπάρχουν άλλα περιθώρια βελτίωσης, δηλαδή ότι το clustering δεν μπορεί να βελτιωθεί χρησιμοποιώντας την αναπαράσταση με τις μειωμένες διαστάσεις. Παρόλα αυτά, όπως αναφέραμε παραπάνω, καταφέραμε να μειώσουμε αισθητά τον χρόνο μέσω της νέας αναπαράστασης και μάλιστα χωρίς να προκύψουν αλλοιώσεις μεταξύ των μετοχών της αρχικής και της νέας, αν συνέβαινε κάτι τέτοιο ίσως η εκτέλεση του clustering για την αναπαράσταση με τις μειωμένες διαστάσεις μας οδηγούσε σε διαφορετική συσταδοποίηση των μετοχών.

Vectors - Reverse Assignment with LSH - Υπολογισμός της μέσης χρονοσειράς ως διάνυσμα

Εκτέλεση αναπαράστασης με 3650d				**Εκτέλεση αναπαράστασης με 1096d**			
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 1	Cluster 2	Cluster 3	Cluster 4
346	1	1	1	346	1	1	1
Clustering time: 0.093685 seconds				**Clustering time:** 0.028162 seconds			
Silhouette				**Silhouette**			
0.972528	1.00	1.00	1.00	0.972504	1.00	1.00	1.00
stotal = 0.972764				stotal = 0.972741			

Παρατηρήσεις

- Συγκρίνοντας τα Cluster time των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/4 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Παράλληλα συγκρίνοντας τις σιλουέτες (stotal) των δύο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, παρατηρούμε ότι και στις 2 περιπτώσεις έχουμε τέλεια σιλουέτα που σημαίνει πως έγινε πολύ καλό clustering. Συμπεραίνουμε, λοιπόν, πως δεν υπάρχουν άλλα περιθώρια βελτίωσης, δηλαδή ότι το clustering δεν μπορεί να βελτιωθεί χρησιμοποιώντας την αναπαράσταση με τις μειωμένες διαστάσεις. Παρόλα αυτά, όπως αναφέραμε παραπάνω, καταφέραμε να μειώσουμε αισθητά τον χρόνο μέσω της νέας αναπαράστασης και μάλιστα χωρίς να προκύψουν αλλοιώσεις μεταξύ των μετοχών της αρχικής και της νέας, αν συνέβαινε κάτι τέτοιο ίσως η εκτέλεση του clustering για την αναπαράσταση με τις μειωμένες διαστάσεις μας οδηγούσε σε διαφορετική συσταδοποίηση των μετοχών.

Time Series - Reverse Assignment with LSH - Υπολογισμός της μέσης χρονοσειράς ως καμπύλη

Εκτέλεση αναπαράστασης με 3650d				**Εκτέλεση αναπαράστασης με 1096d**			
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 1	Cluster 2	Cluster 3	Cluster 4
343	1	2	3	6	1	2	340
Clustering time: 845.630992 seconds				**Clustering time:** 77.455699 seconds			
Silhouette				**Silhouette**			
0.976173	1.00	1.00	0.599784	-0.039987	1.00	0.349308	0.955037
stotal = 0.974152				stotal = 0.934588			

Παρατηρήσεις

- Συγκρίνοντας τα Cluster time των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/11 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Παράλληλα συγκρίνοντας τις σιλουέτες (stotal) των δύο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, παρατηρούμε ότι και στις 2 περιπτώσεις έχουμε τέλεια σιλουέτα που σημαίνει πως έγινε πολύ καλό clustering. Συμπεραίνουμε, λοιπόν, πως δεν υπάρχουν άλλα περιθώρια βελτίωσης, δηλαδή ότι το clustering δεν μπορεί να βελτιωθεί χρησιμοποιώντας την αναπαράσταση με τις μειωμένες διαστάσεις. Παρόλα αυτά, όπως αναφέραμε παραπάνω, καταφέραμε να μειώσουμε αισθητά τον χρόνο μέσω της νέας αναπαράστασης και μάλιστα χωρίς να προκύψουν αλλοιώσεις μεταξύ των μετοχών της αρχικής και της νέας, αν συνέβαινε κάτι τέτοιο ίσως η εκτέλεση του clustering για την αναπαράσταση με τις μειωμένες διαστάσεις μας οδηγούσε σε διαφορετική συσταδοποίηση των μετοχών.

Vectors - Hypercube - Υπολογισμός της μέσης χρονοσειράς ως διάνυσμα

Εκτέλεση αναπαράστασης με 3650d				**Εκτέλεση αναπαράστασης με 1096d**			
Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 1	Cluster 2	Cluster 3	Cluster 4
332	1	11	5	332	1	12	4
Clustering time: 0.143363 seconds				**Clustering time:** 0.042770 seconds			
Silhouette				**Silhouette**			
0.892346	1.00	-0.572443	-0.087252	0.922419	1.00	-0.720200	-0.596337
stotal = 0.832452				stotal = 0.848755			

Παρατηρήσεις

- Συγκρίνοντας τα Cluster time των δυο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, βλέπουμε πως καταφέραμε να πετύχουμε τον πρώτο μας στόχο, μειώνοντας τους χρόνους υπολογισμού στην εκτέλεση που χρησιμοποιεί την αναπαράσταση με διαστάσεις 1095 περίπου στο 1/3 σε σχέση με την εκτέλεση του προγράμματος που χρησιμοποιεί την αρχική αναπαράσταση.
- Παράλληλα συγκρίνοντας τις σιλουέτες (stotal) των δύο αυτών εκτελέσεων με την χρήση των δύο διαφορετικών αναπαραστάσεων, παρατηρούμε ότι και στις 2 περιπτώσεις έχουμε τέλεια σιλουέτα που σημαίνει πως έγινε πολύ καλό clustering. Συμπεραίνουμε, λοιπόν, πως δεν υπάρχουν άλλα περιθώρια βελτίωσης, δηλαδή ότι το clustering δεν μπορεί να βελτιωθεί χρησιμοποιώντας την αναπαράσταση με τις μειωμένες διαστάσεις. Παρόλα αυτά, όπως αναφέραμε παραπάνω, καταφέραμε να μειώσουμε αισθητά τον χρόνο μέσω της νέας αναπαράστασης και μάλιστα χωρίς να προκύψουν αλλοιώσεις μεταξύ των μετοχών της αρχικής και της νέας, αν συνέβαινε κάτι τέτοιο ίσως η εκτέλεση του clustering για την αναπαράσταση με τις μειωμένες διαστάσεις μας οδηγούσε σε διαφορετική συσταδοποίηση των μετοχών.