3η Εργασία 2021 - 2022

Συνεργάτες:

- **Βησσαρίων Μουτάφης , A.M:1115201800119, mail: sdi1800119@di.uoa.gr**
- **Αρίστη Παπασταύρου , A.M:1115201800154, mail: sdi1800154@di.uoa.gr**

1. Installation and Run

1.1 Prerequisites

To install prerequisites for this project please run the following command

```
~$ pip3 install requirements.txt
```

1.2 Run

Για το compilation και run της εφαρμογής παρέχουμε ένα Makefile στο root directory. Οι πιθανές εντολές είναι:

```
~$ python3 forecast.py -n [number of selected timeseries] -d [dataset path] [-h]
~$ python3 detect.py -d [dataset] -n [number of time series selected] -mae [error value as double] [-h]
~$ python3 reduce.py  d  [datase] -q [queryset] -od [output_dataset_file] -oq [output_query_file] [-h]
```

1.3 Training vs Normal Run

Το default run behaviour των εκτελέσιμων βασίζεται σε μοντέλα τα οποία βρίσκονται είτε στο default path './', είτε σε κάποιο άλλο path το οποίο θα διευκρινιστεί με την προσθήκη του flag, '–model-path' και το path του directory που βρίσκεται το αντίστοιχο μοντέλο.

Για την εκπαίδευση των μοντέλων τα εκτελέσιμα δέχονται τις ίδιες παραμέτρους, με την προσθήκη του flag '–train' κατα το τρέξιμο. Για την ρύθμιση των υπερπαραμέτρων χρησιμοποιούμε τους configuration files μέσα στο directory './config/', όπου έχουν διακριτά ονόματα για κάθε ένα από τα executables.

1.4 Default Models

Τα models που δημιουργούνται μετά την εκπαίδευση σώζονται είτε στο default path './' είτε σε directory path δοσμένο από τον χρήστη με την προσθήκη του παραπάνω cmd arg '–model-path'. Συγκεκριμένα:

- Για το forecasting σώζουμε ένα μοντέλο με όνομα **'forecast'**
- Για το outlier detection σώζουμε ένα μοντέλο με όνομα **'detect'**
- Για το dimensionality reduction, σώζουμε το μοντέλο του encoder μόνο, με όνομα 'reduce'

Παρέχουμε 3 μοντέλα, 1 για κάθε ερώτημα στο directory trained-models, τα οποία έχουν εκπαιδευτεί με τα hyper-parameters, που έχουμε αφήσει ως default στο directory config/, για κάθε εκτελέσιμο.

2. Κατάλογος αρχείων πηγαίου κώδικα

Στο root directory θα βρείτε τα αρχεία/καταλόγους:

- src/
- misc/
- config/
- forecast.py
- detect.py
- reduce.py

Στο directory src θα βρείτε τα αρχεία:

- TimeSeriesForecastingFramework.py
- TimeSeriesCNNAutoEncoderFramework.py
- TimeSeriesAutoEncoderFramework.py
- ArgParser.py
- utilities.py

Στο directory config θα βρείτε τα αρχεία-configurations:

- forecast_config.py
- detect_config.py
- reduce_config.py

Στο directory misc θα βρείτε τα αρχεία .ipynb στις πρώτες εκδόσεις τους:

- AutoEncoder.ipynb
- Project3_A1.ipynb
- Project3_C-final.ipynb
- Project3_C-Rev.ipynb
- Project3_C.ipynb

3. Dataset preprocessing

3.1 Forecasting

Για το forecasting, τα δεδομένα μας πρέπει να είναι σε μορφή timseries-segments με μήκος, lookback, που αντικατοπτρίζει τα πόσα timesteps συμπεριλαμβάνουμε per sequence. Αυτό υλοποιεί η συνάρτηση preprocess_timeseries, η οποία δέχεται ενα timeseries, εφαρμόζει min-max normalization και επιστρέφει τα επιθυμητα X, y και τα max και min. Η εφαρμογή min max normalization γίνεται παντα ανά χρονοσειρά και κατά το plotting έχουμε φροντίσει να κρατήσουμε τα min, max statistics της αντίστοιχης χρονοσείρας για να εφαρμόσουμε inverse normalization.

3.2 Outlier Detection

Με όμοιο τρόπο με το forecasting, (όδων αναφορά την μορφή των δεδομένων μας), υλοποιήσαμε την συνάρτηση create_dataset η οποία εκτελει mean standardization πάνω στο input time-series καθώς επιστρέφει τα επιθυμητα X, y και mean,sigma. Όπως και πριν η εφαρμογή του mean standardization γίνεται παντα ανά χρονοσειρά και κατά το plotting έχουμε φροντίσει να κρατήσουμε τα μεαν, σιγμα statistics της αντίστοιχης χρονοσειράς για να εφαρμόσουμε inverse normalization (αντί να χρησιμοποιήσουμε τον ήδη υπάρχοντα scaler της sklearn).

3.3 Dimensionality Reduction

Το normalization είναι ίδιο με το Forecasting με την διαφορά πως ο πίνακας y ειναι ίδιος με τον πίνακα X μιας και που ο autoencoder προσπαθεί να φτιάξει την original timeseries από την latent timeseries, οπότε το μέτρο σύγκρισης καταλήγει να είναι η ομοιότητα των 2 χρονοσειρών.

3.4 Model Wrappers

Τα Model Wrappers για κάθε ενα από τα προβλήματα αποτελούν το module του model που χρησιμοποιούμαι για την λύση τους. Η δομή του μοντέλου καθώς και άλλες hyperparameters καθορίζονται κατά την κατασκευή του και συγκεκριμένα παρέχονται στον αντίστοιχο config φακελό, με τον οποίο προμηθεύουμε το κάθε εκτελέσιμο. Τα μοντέλα εκπαιδεύονται, δεδομένου ενος dataset το οποίο χωρίζουν σε train-test και χρησιμοποιούν early stopping ώστε να σώσουν χρόνο αλλά και να αποφύγουν το overfitting.

3.5 Problem Wrappers

Οι problem wrappers, μαζεύουν τα tasks του dataset preprocessing, dataset assignment, model fitting, evaluation και plotting σε μια κλάσση η οποία παρέχει 3 βασικές μεθόδους: ___init___, solve, plot_graphs και στην περίπτωση του Γ, reduce and export. Οι μεοδοι αυτοί λύνουν ενα πρόβλημα δεδομένου ενός model και ενός dataset, το τελευταίο εκ των οποίων, περνούν από preprocessing και το χωρίζουν σε test-train sets, για κάθε timseries, κρατώντας τα κατάλληλα στατιστικά για το scaling. Η μέθοδος plot graphs, δίνειτην δυνατότητα στον χρήστη να εκτυπώσει επιλεγμένα ή και όλα τα plots για τα predictions και τα true timeseries ώστε να κάνει visualize τα αποτελέσματα του model.

4. Time Series - Forecasting with LSTMs

Το model αποτελείτε από μια λίστα LSTMs layers τα οποία έχουν τον δικό τους αριθμό units. Για να αποφύγουμε το overfitting δίνουμε είτε balanced αριθμό layers, είτε αρκετά μεγάλο training dataset ρυθμίζοντας σωστα το dropout καθε φορά.

4.1 Lookback-LSTM units Relationship

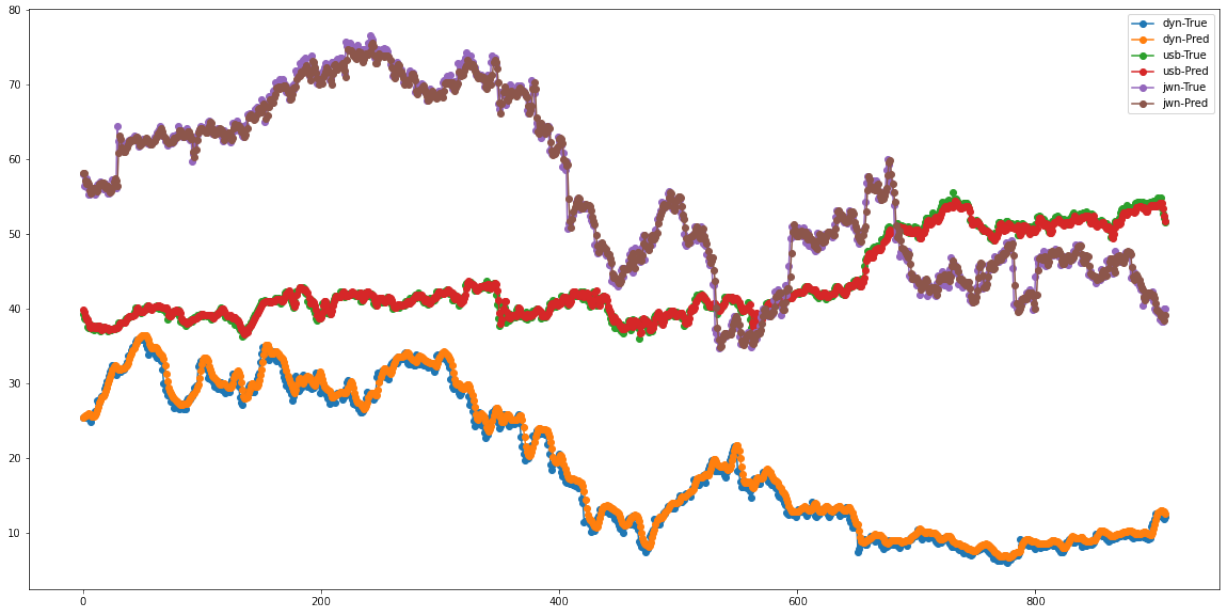
Παρατηρούμε πως όσο αποκλείνει το lookback length με τον αριθμό των hiden layers των lstm layers το μοντέλο θα συγκλίνει πολύ πιο αργά, με κρίσιμο σημείο την σχέση $\#units > 2 \times lookback$, όπου το μοντέλο καταλήγει να περνάει από αρκετά hidden states, χωρίς input sequences, εφόσον το ποσοστό των ουσιαστικών sequences ειναι πολύ μικρό ποσοστο της ακολουθίας που εισέρχεται στο LSTM. Σε αυτή την περίπτωση μπορούμε να πετύχουμε μεγάλο prediction score, αλλά με μεγάλο υπολογιστικό κόστος.

Από την άλλη αν το lookback ειναι αρκετά μεγαλύτερο από τα περισσότερα hiden units του τελευταίου layer, τοτε το μοντέλο πιάνει την γενική συμπεριφορά των timeseries έχοντας όμως σημαντική επίπτωση στα predicted values. Το ενδιαφέρον εδώ ειναι να παρατηρήσουμε πως γνωρίζοντας την συμπεριφορά (πως κινείται η μετοχή) είναι πιο υπολογιστικά φθηνό και ειναι συχνά αρκετό για την πρόβλεψη μιας προσσέγιστικής τιμής.

4.2 Experimental results

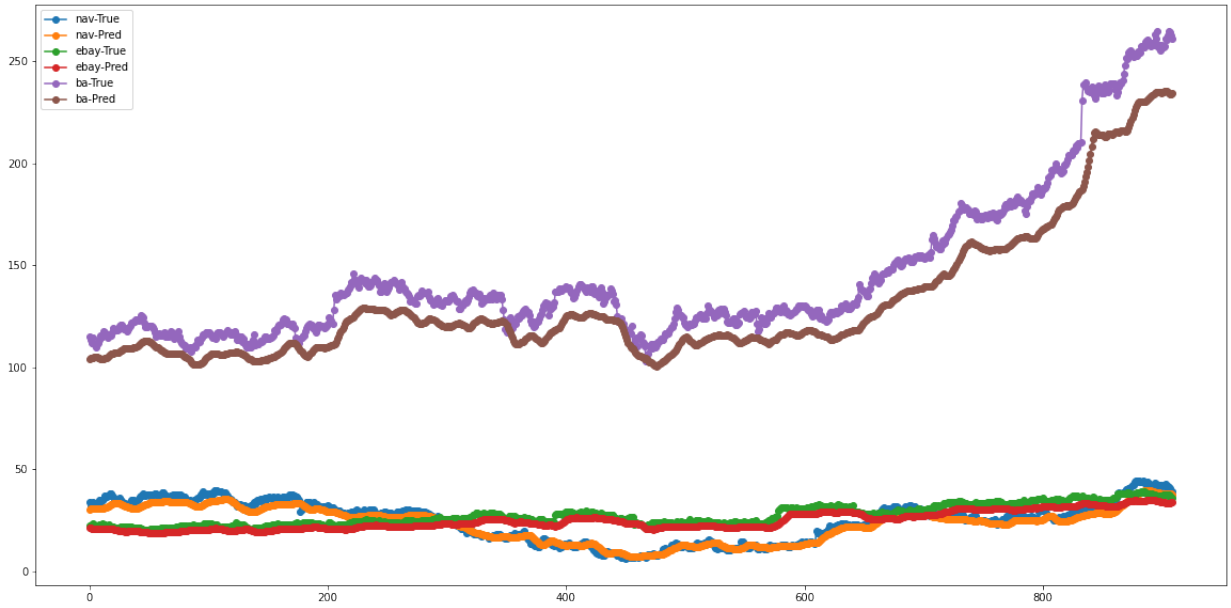
Κάνοντας ορισμένα πειράματα πάνω στο ιδιο dataset με 50 timeseries από το nasdaq2007_17.csv κάναμε προβλέψεις για τις πρώτες 3 χρονοσειρές του dataset. Το evaluation πραγματοποιήθηκε στο 20% του κάθε timseries το οποίο πάρθηκε από το τέλος, ώστε να πραγματοποιήσουμε μια realistic πρόβλεψη για την τιμή, σε πραγματικές συνθήκες.
***Test* − 1**

- Lookback: 10
- Epochs : 50
- Batch Size: 1024
- Layers-Units = [10, 10, 10]
- dropout: 0.1
- loss: MSE (final loss value under 0.01)


dyn-True
dyn-Pred
usb-True
usb-Pred
jwn-True
jwn-Pred
80
70
60
50
40
30
20
10
0
200
400
600
800

Test − 2

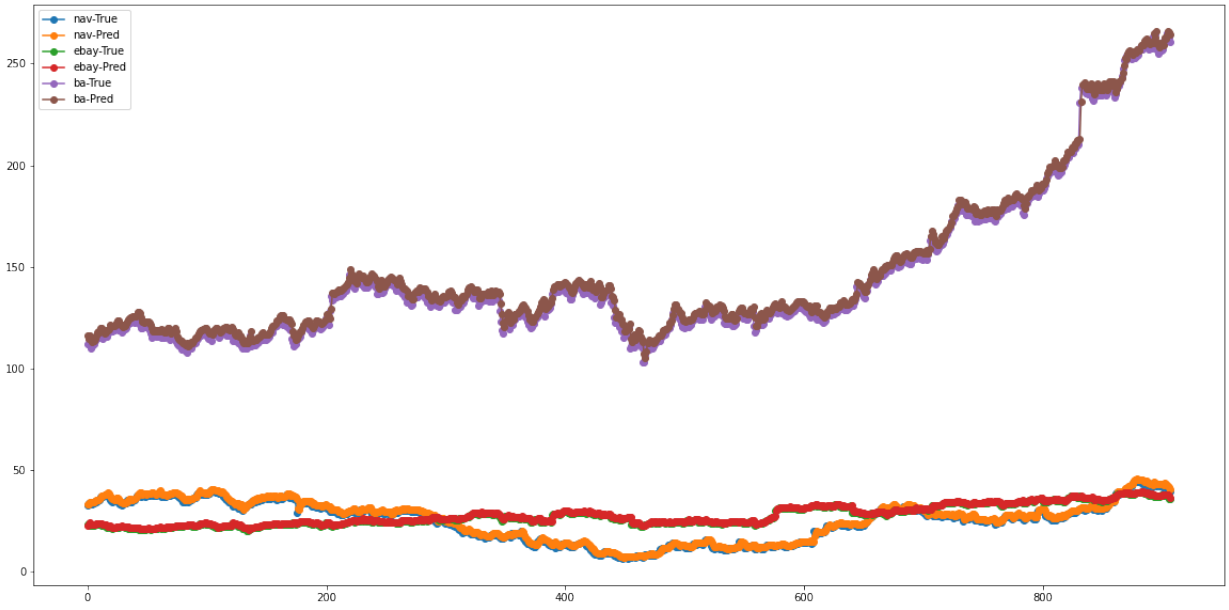
- Lookback: 10
- Epochs : 50
- Batch Size: 1024
- Layers-Units = [10, 10, 10]
- dropout: 0.5
- loss: MSE (final loss under 0.02)



Παρατηρούμε πως η αύξηση του dropout πανω από μια συγκεκριμένη τιμή (πειραματικα ηταν 0.05) προκαλεί underfit στο μοντέλο το οποίο λογω του αριθμού των δεδομένων χρονοσειρών δεν παθαινει εύκολα overfit και γενικεύει πολύ καλα.

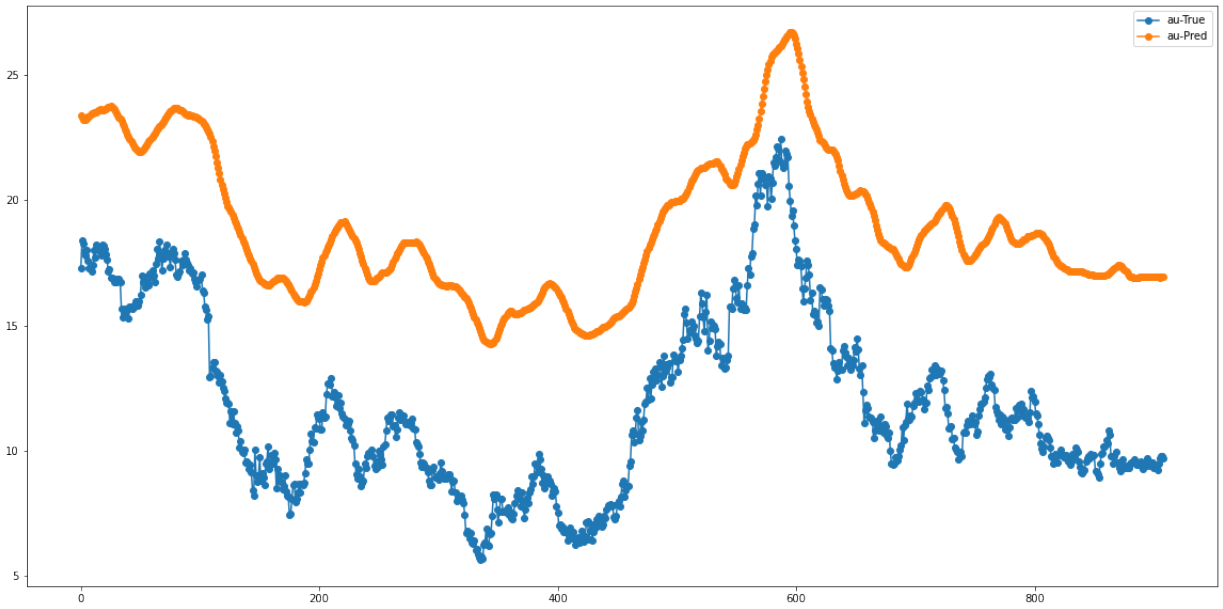
Test – 3

- Lookback: 20
- Epochs : 50
- Batch Size: 1024
- Layers-Units = [20, 20, 30]
- dropout: 0.0
- loss: MSE (final loss value under 0.005)



Test-4 with only 1 timeseries in the training dataset

- Lookback: 15
- Epochs : 50 (convergence in only 2sec, faster than all)
- Batch Size: 64
- Layers-Units = [5, 5, 5]
- dropout: 0.05
- loss: MSE (final loss value under 0.01)



Βλέπουμε πως η predicted γραμμή δεν είναι αρκετά ακριβής, ως προς την τιμή της μετοχής, όμως πιάνει την διακύμανση της κατά την πάροδο του χρόνου σωστα, κάτι το οποίο είναι χρήσιμο όταν μας ενδιαφέρει κάτι πιο γενικό από την τιμή όπως *πότε θα ανέβει η τιμή της μετοχής*.

5. Time Series - Outlier Detection with LSTM Autoencoder

Το model αποτελείται από μία ακολουθία LSTM layers τα οποία έχουν δικό τους αριθμό από units. Μέσα στον autoencoder θα βρείτε μία λούπα encoding όπου διασχίζοντας τα lstm layers που δόθηκαν από τον χρήστη χτίζουμε και το ανάλογο lstm, ενώ για το decoding έχουμε ένα loop το οποίο διασχίζει με ανεστραμμενη σειρα τα lstm layers που δόθηκαν ως εισοδος.Όπως μας συμβουλεύει και το αντίστοιχο άρθρο, στο τέλος του encoder loop, δημιουργούμε ένα διπλότυπο του τελευταίου LSTM layer, για να πάρουμε ένα 2-d array για το επόμενο layer του decoder loop.Μετά την υλοποίηση της κλάσης του μοντέλου μας, πειραματιστήκαμε αρκετά με optimizations πάνω στην διάταξη της κλάσης και με το αντίστοιχο tuning.Ενδεικτικά:

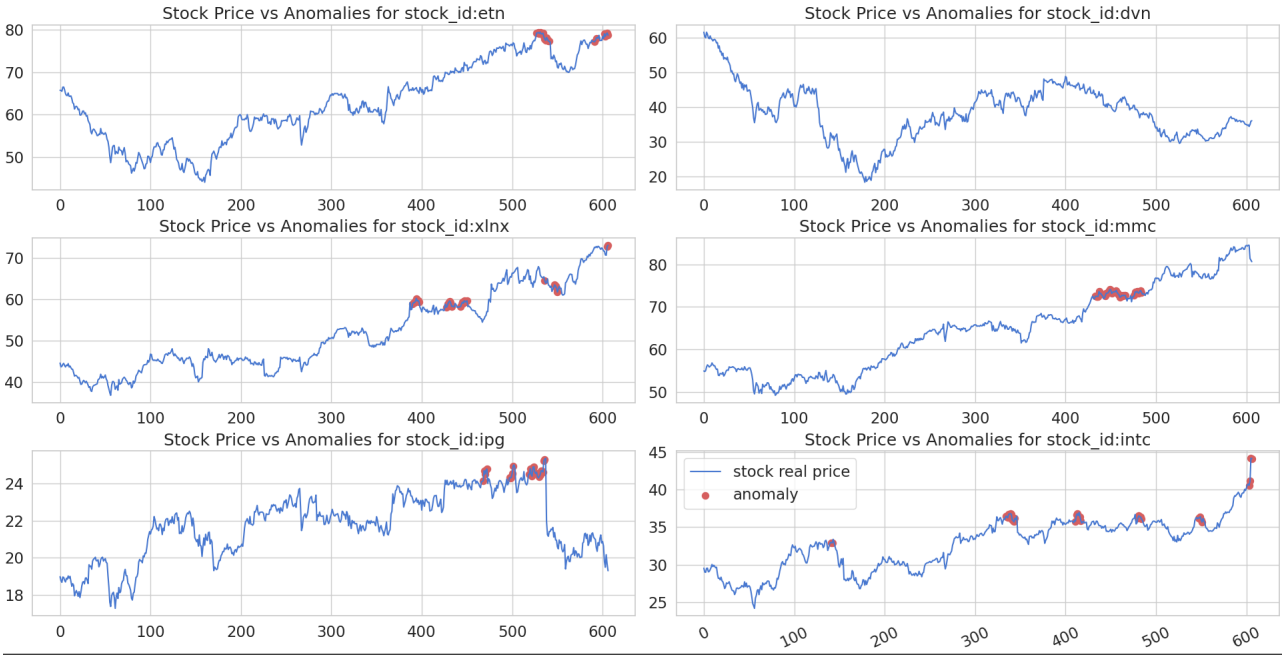
- Dropout layers μεταξύ κάθε lstm layer του stack
- δοκιμάσαμε να βάλουμε σαν πάράμετρο του κάθε lstm, activation functions όπως tanh,sigmoid,RelU αλλά τελικά η default αψτιωατιον του LSH (tanh), αποδίδει καλύτερα.
- Βάλαμε BatchNormalisation μεταξύ των layers, το οποίο τελικά βοήθησε στην ορθότερη κλιμακωτή μείωση του error.
- Βάλαμε σαν παράμετρο στο κάθε LSTM, batch_input_shape για optimised batching.
- Δοκιμάσαμε διαφορετικούς optimizers στο compilation του μοντέλου αλλά τελικά ο adam ειχε τα καλύτερα αποτελέσματα.

Η σχέση μεταξύ Lookback και lstm layers είναι όμοια με αυτή που περιγράφηκε παραπάνω.Αξίζει να σημειωθεί πως σημαντική διαφορά στην απόδοση προκύπτει και από τον αριθμό και αξία των lstm layers. Συγκεκριμένα με αύξηση των lstm layers έχουμε σημαντική μείωση στο test loss, ενώ όταν τα layers είναι δυνάμεις του 2 και μεγαλύτερα σε αξία (πχ 64, 100 ...) έχουμε σημαντική αύξηση στον χρόνο εκτέλεσης αλλά πολύ μικρότερο τελικό test loss και καλύτερο fit του μοντέλου μας.

5.1 Experimental results

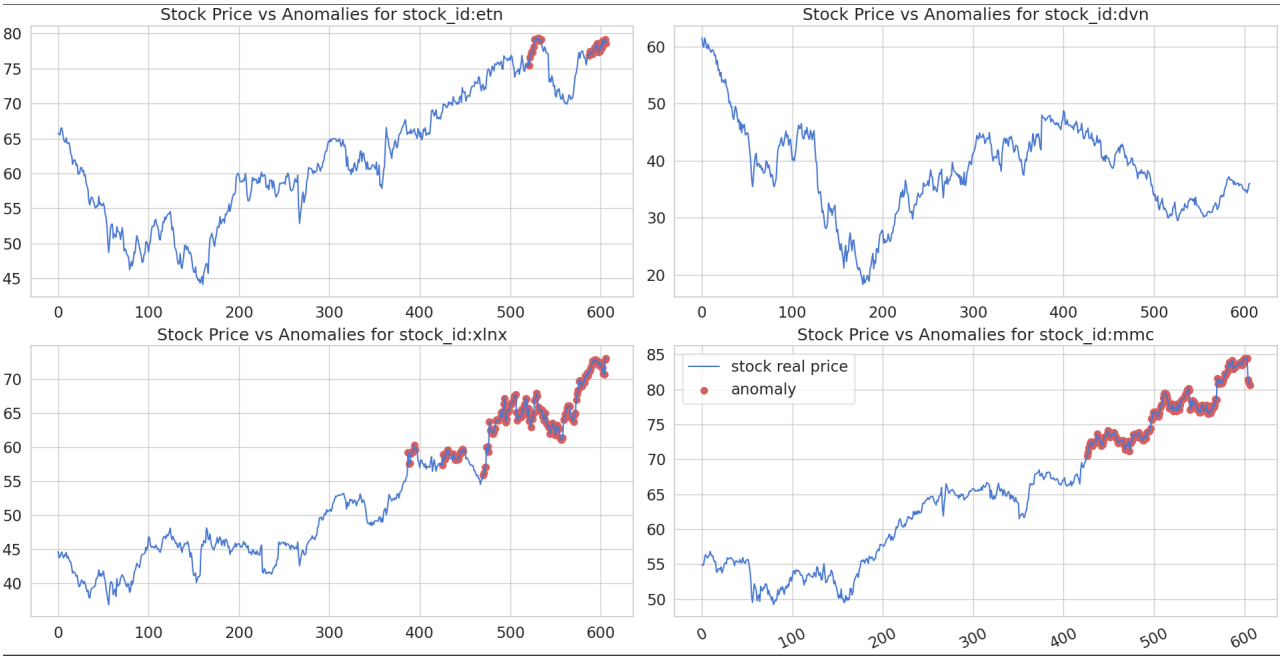
Test-1 with 50 timeseries in the training dataset

- Lookback: 10
- Epochs : 25 (convergence achieved in 14 epochs(better results than the rest of the experiments but slower.Each epoch takes about 50 seconds to execute completely))
- Batch Size: 256
- Layers-Units = [100, 64, 64]
- dropout: 0.2
- loss: MAE (final loss value under 0.01)
- mae threshold for plot of anomalies: 0.5



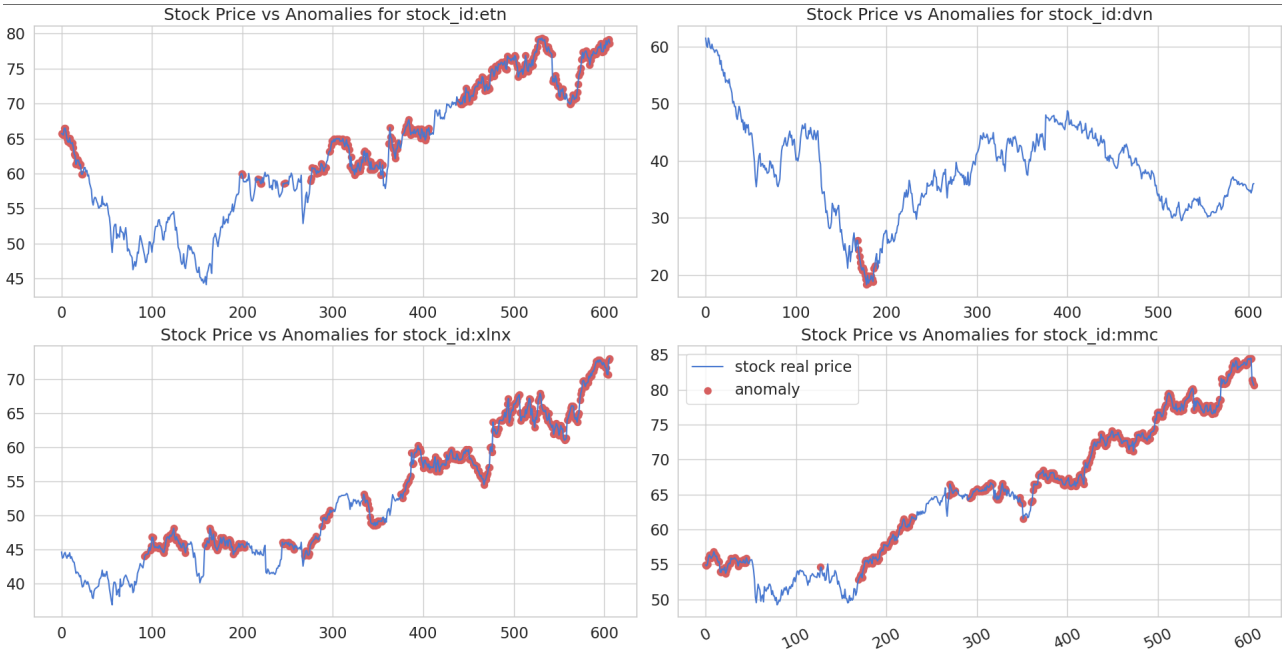
Test-2 with 10 timeseries in the training dataset

- Lookback: 10
- Epochs : 25 (convergence achieved in 20 epochs)
- Batch Size: 256
- Layers-Units = [10,10,10,10,20]
- dropout: 0.1
- loss: MAE (final loss value under 0.11)
- mae threshold for plot of anomalies: 0.5



Test-3 with 10 timeseries in the training dataset

- Lookback: 10
- Epochs : 25 (convergence achieved in 20 epochs)
- Batch Size: 256
- Layers-Units = [10,10,10,10,20]
- dropout: 0.3
- loss: MAE (final loss value over 0.3)
- mae threshold for plot of anomalies: 0.5

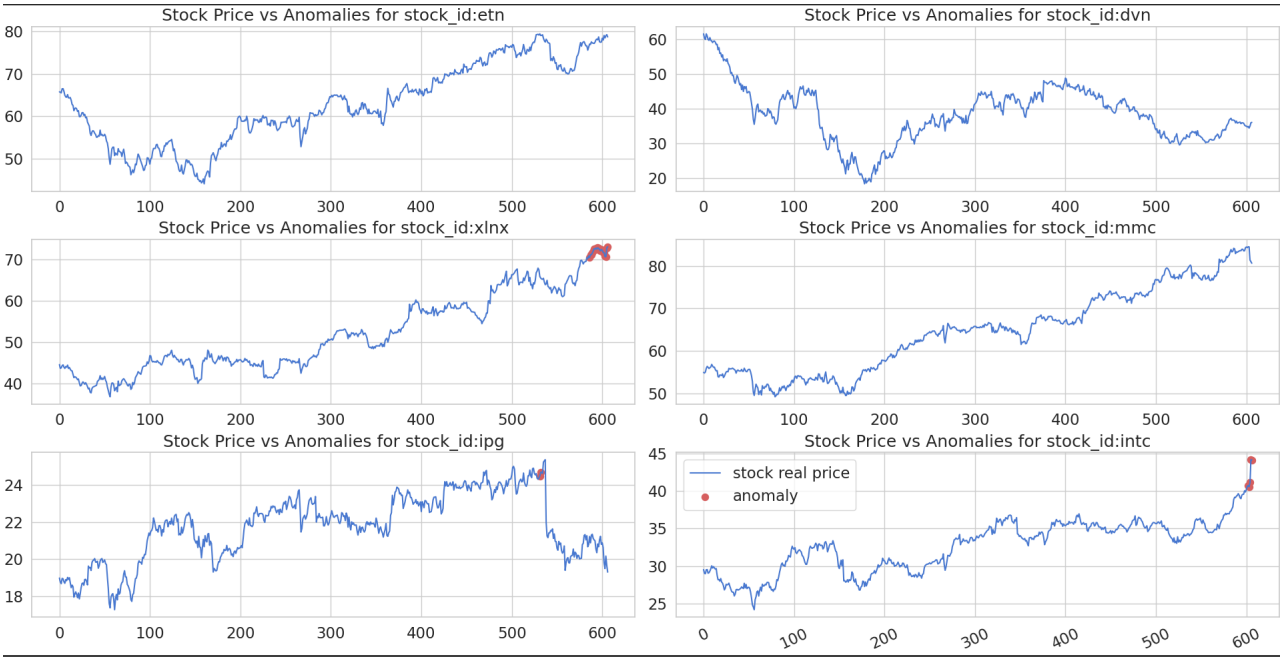


Παρατηρούμε και από τα γραφήματα αλλά και από το τελικό loss του training ότι όσο αυξάνοθμε το dropout τότε τόσο αυξάνεται το loss και τόσα περισσότερα anomalies έχουμε σε κάθε timeseries. Οπότε γενικά για καλύτερο training προτιμούμε

$$dropout < 0.3$$

Test-4 with 50 timeseries in the training dataset

- Lookback: 10
- Epochs : 25 (convergence achieved in 18 epochs)
- Batch Size: 256
- Layers-Units = [100, 64, 64, 36, 36]
- dropout: 0.1
- loss: MAE (final loss value over 0.3)
- mae threshold for plot of anomalies: 0.5



6. Time Series - Dimensionality Reduction with CNN Autoencoder

6.1 Operational and Architectural Details

Τα models για την κατασκευή του autoencoder καθορίζονται βάσει της αρχιτεκτονικης του encoder την οποια παρέχουμε μέσω της μεταβλητής, CNN_LAYER_SETTINGS, την οποία μπορείτε να αλλάξετε στο κατάλληλο αρχείο config. Η βασική αρχή για τον autoencoder ειναι να χρησιμοποιούμε όλα τα layers που δίνει ο χρήστης αρκεί η διάσταση του sequence να μην πέφτει κάτω απο το latent dimension ούτε να ανεβαίνει πάνω από το original dimension, αντίστοιχα στους encoder και decoder. Αν ο χρήστης παρέχει CNN layers με τέτοιο τρόπο ώστε να μην μπορέι να σχηματιστεί το latent dimension, χρησιμοποιούμε Linear Dense Layers, αφου κάνουμε Flatten το output και ύστερα χρησιμοποιούμε reshape για να δώσουμε το latent dimension. Ίδια λογική ακολουθούμε και κατά το reconstruction στον decoder.

Η επίλυση του προβλήματος και η εκτύπωση αποτελεσμάτων έχει ίδια λειτουργικότητα με τα προηγούμενα προβλήματα.

6.2 Encoder Usage

Για την χρήση του εκπαιδευμένου encoder παρέχουμε τις συναρτήσεις reduce and export, που επιστρέφει ενα DataFrame με όλες τις complexity reduced timeseries, και την create compressed file που γράφει τον τελικό φάκελο, σύμφωνα με την ονομασία που παρείχε ο χρήστης.

7. Experimental Notes

Παρόμοια με το Β, ο autoencoder προβλέπει καλύτερα όσο πιο κοντα ειναι ο latent space στον original space. Παρ όλα αυτά με τις σωστές ρυθμίσεις των φιλτρων μπορούμε να καλύψουμε το κενό αυτό. Το πιο σημαντικό ομως ειναι η παροχή μεγάλου αριθμού timeseries για το training αφού πειραματικά μπορεί κανείς να δει πως για την σωστή ανακατασκευή των μετοχών πρεπει να πάρουμε τουλάχιστον 20 μετοχες, ώστε ο autoencoder να εκπαιδευθεί σε ικανοποιητικό βαθμό. Όπως και στα προηγούμενα ερωτήματα το dropout βοηθάει μόνο οταν είναι κάτω από ενα threshold (περίπου 0.07), ώστε να μην αλλιώνει υπερβολικά την εικόνα των timeseries, ειδικά αυτών που η διακύμανση ειναι αρκετά μικρή (πχ. standard dev = .5 εως και 2 μονάδες), αφού οι μικρές αλλαγές και τα patterns είναι ανεπαισθητα υπό την επήρεια μεγάλου dropout.

8. Time Series - Clustering Results with Reduced Dimensions

Στα .zip folder θα βρείτε και ένα folder(Metrics) που περιέχει τις μετρήσεις που κάναμε και με κανονικών-διαστάσεων timeseries αλλά και με compressed inputs, έτσι ώστε να μπορέσουμε να συγκρίνουμε τις μεθόδους μεταξύ τους.*Όποιο αρχείο έχει στο όνομα του την λέξη comp πρόκειται για μετρήσεις πάνω σε encoded timeseries, αλλιώς μιλάμε για original dimensions.* Η συγκριση μετρήσεων ως προς την απόδοση της 2ης εργασίας για τα exported_(reduced from encoder) timeseries, αλλά και τα timeries κανονικών διαστάσεων όπως μας δόθηκαν, θα βασιστούν ως προς την χρονική απόδοση και το accuracy των αποτελεσμάτων.

8.1 Χρονική απόδοση:

Όπως ήταν αναμενόμενο, σε όποια μέθοδο και αν δοκιμάσαμε (είτε πρόκειται για clustering ή για το κομμάτι του search) είχαμε σημαντική μείωση χρόνου η οποία έφτανε σε ορισμένες περιπτώσεις και σε υποδεκαπλάσιο χρόνο συγκρισιακά με τον εκτελέσιμο χρόνο των αυθεντικών timeseries.Συγκεκριμένα στο **search, χρησιμοποιώντας την μετρική Discrete - Frechet** για 60 timeseries αντίστοιχα έχουμε:

L = 5, k = 7, encoded timeseries

```
tApproximateAverage: 11200.9ms
tTrueAverage: 3661.87ms

Total Stats:
Average per point accuracy: 1
Average brute_KNN/approx_KNN time ratio: 0.326962
Average approx_KNN dist / true_KNN dist error: -nan (δεν ορίζεται εφόσων και
↪  οι 2 όροι είναι ίσοι με 0)
```

L = 5, k = 7, non-encoded timeseries

```
tApproximateAverage: 61070.7ms
tTrueAverage: 16733.6ms

Total Stats:
Average per point accuracy: 1
Average brute_KNN/approx_KNN time ratio: 0.460887
Average approx_KNN dist / true_KNN dist error: -nan (δεν ορίζεται εφόσων και
↪  οι 2 όροι είναι ίσοι με 0)
```

Στο παραπάνω παράδειγμα έχουμε 10% καλύτερη απόδοση όσο αναφορά το Average brute_KNN/approx_KNN time ratio και περίπου 6 φορές μικρότερο χρόνο όταν χρησιμοποιούμε τα encoded timeseries.
Με τις μετρικές MAF και Average approx_KNN dist / true_KNN dist error δεν αξίζει να ασχοληθούμε μιας και δεδομένου ότι δώσαμε σαν input και query το ίδιο file, τα distanceAproximate και distanceTrue θα είναι 0 σε κάθε query για τον αλγόριθμο discrete frechet.
Αν χρησιμοποιουσαμε το nasd_input.csv και nasd_query.csv (και τα αντιστοιχα encoded reduced files τους) θα παιρναμε κατι τετοιο σε στατιστικα:

L = 5, k = 3, encoded timeseries

```
tApproximateAverage: 177.896ms
tTrueAverage: 257.88ms
MAF: 1.96753

Total Stats:
Average per point accuracy: 0.9
```

```
Average brute_KNN/approx_KNN time ratio: 1.51114
Average approx_KNN dist / true_KNN dist error: 0.0967532
```

L = 5, k = 3, non-encoded timeseries

```
tApproximateAverage: 933.614ms
tTrueAverage: 1269ms
MAF: 1.35179

Total Stats:
Average per point accuracy: 0.8
Average brute_KNN/approx_KNN time ratio: 1.54071
Average approx_KNN dist / true_KNN dist error: 0.0422816
```

Καταλήγουμε λοιπόν πως τα MAF, accuracy, dist error κτλπ, δεν έχουν εγκυρότητα μεταξύ των 2 dataset διότι το compression και η προβολή σε χαμηλότερο complexity επηρεάζει αρκετά την απόσταση των καμπυλών, οχι όμως και σε βαθμό που ο πραγματικός nearest neighbour να είναι εκτός των top 5 nearest neighbours (συμφωνα με πειραματικά τρεξίματα που κάναμε). Για να βελτιώσουμε το accuracy θα πρεπει να ανεβάσουμε την latent dimension κατα το reducing, ετσι ώστε το παραγόμενο dataset να είναι πιο κοντά στο original dataset. Συγκεκριμένα, έχουμε μείωση complexity της τάξης $\lceil complexity \rceil lookback \times (original\_dim - latent\_dim)$, εφόσον χωρίζουμε την χρονοσειρά σε κομμάτια μήκους lookback, χωρίς να συμπίπτουν και μειώνουμε την διάσταση της κάθε υπό-χρονοσειράς από original dim, η οποία είναι ίδια με την τιμή του lookback, σε latent dim. Μπορεί, λοιπόν, κανείς να καταλάβει, πως η μείωση των features της χρονοσειράς επιφέρει σημαντική φθορά της πληροφορίας, όμως με την προβολή σε ενα latent dim που εξισσοροπεί το accuracy και το speedup μπορούμε να πετύχουμε ακριβή αποτελέσματα, σε ικανοποιητικό χρονικό διάστημα.

Clustering Αντίστοιχα με το search και στο clustering έχουμε σημαντικά μειωμένους χρόνους όταν χρησιμοποιούμε encoded timeseries. Μπορείτε να δείτε λεπτομερώς τα αποτελέσματα στο folder με τα outputs αλλά κάποια από τα αποτελέσματα μας παρέχονται και εδώ για καλύτερη οπτικοποιήση των παρατηρήσεων μας.

Χρόνοι εκτέλεσης για encoded reduced timeseries με 4 clusters

k/L	3	5	7
3	57.2073	64.8086	4122.69
4	59.9415	91.3419	100.638
5	47.2672	128.593	119.778

Χρόνοι εκτέλεσης για original timeseries με 4 clusters

k/L	3	5	7
3	141.151	353.347	4122.69
4	386.701	91.3419	100.638
5	272.065	128.593	119.778

Αξιοσημείωτο είναι επίσης το γεγονός ότι έαν δώσουμε στην 2η εργασία ως inputs όλο το dataset σε compressed μορφή θα καταφέρει να βγάλει αποτέλεσμα σε **237.083s** και η Silhouette αν και χρονοβόρα καταφέρνει να τυπώσει αποτέλεσμα μεσα σε 30 λεπτά.Εάν όμως δώσουμε στην εργασία το dataset στην κανονική του μορφή, θα καταφέρει να βγει αποτέλεσμα σε 756 s αλλά η Silhouette ακόμα και μετά απο 4 ώρες εκτέλεσης ακόμα δεν είχε καταφέρει να τυπώσει αποτέλεσμα.
Στο folder με τα outputs θα βρείτε μετρήσεισγια όλες τις μεθόδους που αναπτύξαμε στην 2η εργασία,χρησιμοποιώντας κάθε φορά διαφορετικό tuning, για να πάρουμε εύρος αποτελεσμάτων. Ακόμη κάναμε και κάποια πειράματα με το Continuous Frechet, όμως επειδή εξαρχής ήταν αρκετά αργος σαν αλγόριθμος, μπορούμε να παρατηρήσουμε πόσο καλύτερη χρονική απόδοση έχουμε με τα encoded timeseries αφού για input/query αρχείο 10 encoded timeseries είχαμε παρόμοιο χρόνο εκτέλεσης με input/query αρχείο 5 original timeseries

8.2 Accuracy αποτελεσμάτων:

Σχετικά με την ακρίβεια των αποτελεσμάτων: Συμπεράσματα ως προς το accuracy μπορούμε να βγάλουμε μόνο από τα διάφορα τρεξίματα πάνω στο cluster, μιας και στο search, δεδομένου ότι δίνουμε το ίδιο αρχείο ως input αλλά και ως query, έχουμε παντού 100% accuracy στην εύρεση k-nearest neighbors.
Παρατηρώντας το output της silhouette είτε χρησιμοποιώντας ως update το Mean_Curve ή το Mean_Vector και οποιοδήποτε assignment (LSH|HyperCube|LSH Frechet) αν και σε πρώτη φάση φαίνεται να έχουν κοντινούς μέσους όρους silhouette(λίγο καλύτερους όταν έχουμε reduced-encoded timeseries), αν προσέξουμε περισσότερο θα δούμε ότι χρησιμοποιώντας reduced-encoded timeseries έχουμε πολύ καλύτερο διαμοιρασμό curves σε clusters (για το δοθέν dataset) σε σχέση με τα original timeseries.Στα περισσότερα παραδείγματα βλέπουμε ότι τα original curves έχουν κάποια από τα clusters χωρίς ανατεθημένα curves ή μπορεί να παρουσιάζουν και αρνητικό silhouette result. Ενώ με την χρήση reduced-encoded timeseries μπορεί να παρατηρήσουμε κάποια clusters με αρνητικό silhouette evaluation result αλλά κατα κύριο λόγο όλα τα clusters έχουν curves που τους έχουν ανατεθεί και ως επι των πλείστων παρουσιάζουν θετικό silhouette score.Παρακάτω παραθέτουμε κάποια πινακάκια και γραφήματα για καλύτερη οπτικοποίηση των αποτελεσμάτων.

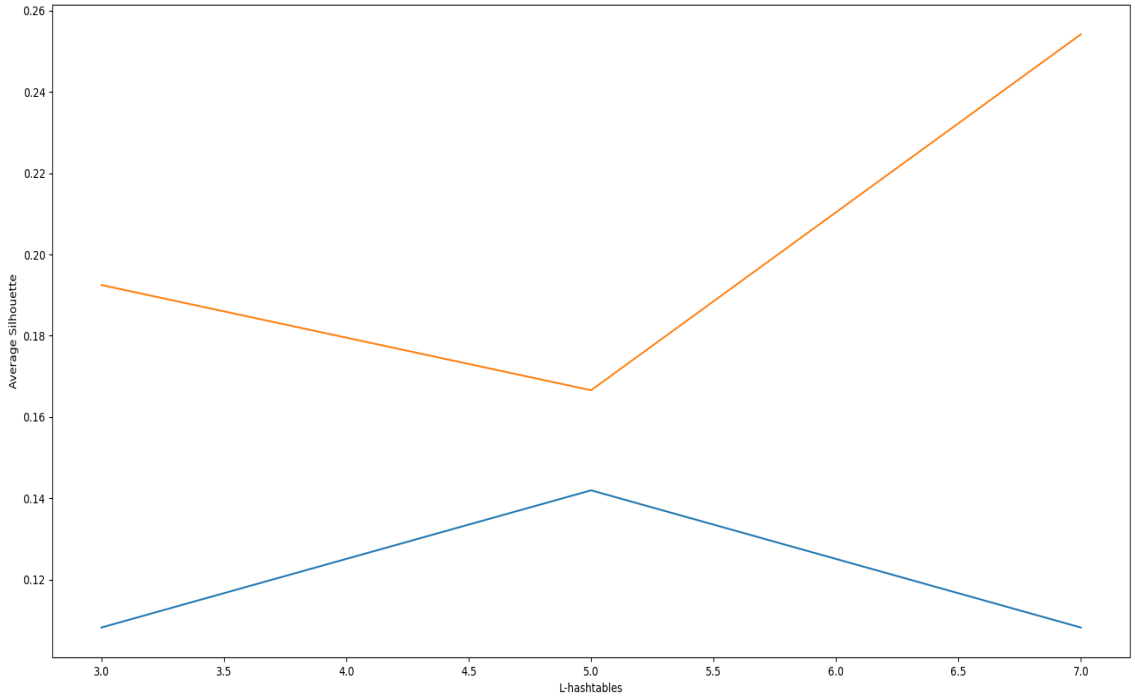
Silhouette results for encoded reduced timeseries with 4 clusters using LSH Frechet

k/L	3	5	7
3	[0.066,-0.111,0.095,0.347,0.099]	[0.112,0.125,0.435,0.144,0.204]	[-0.041,0.476,0.183,0.072,0.173]
4	[0.426,0.036,0.122,-0.007,0.144]	[0.50,-0.032,0.124,-0.001,0.148]	[0.134,0.152,0.112, 0.455,0.213]
5	[0.439,0.036,-0.050,0.12,0.134]	[-0.059,0.233,0.106,0.107,0.097]	[0.102,0.091,0.309,-0.047, 0.114]

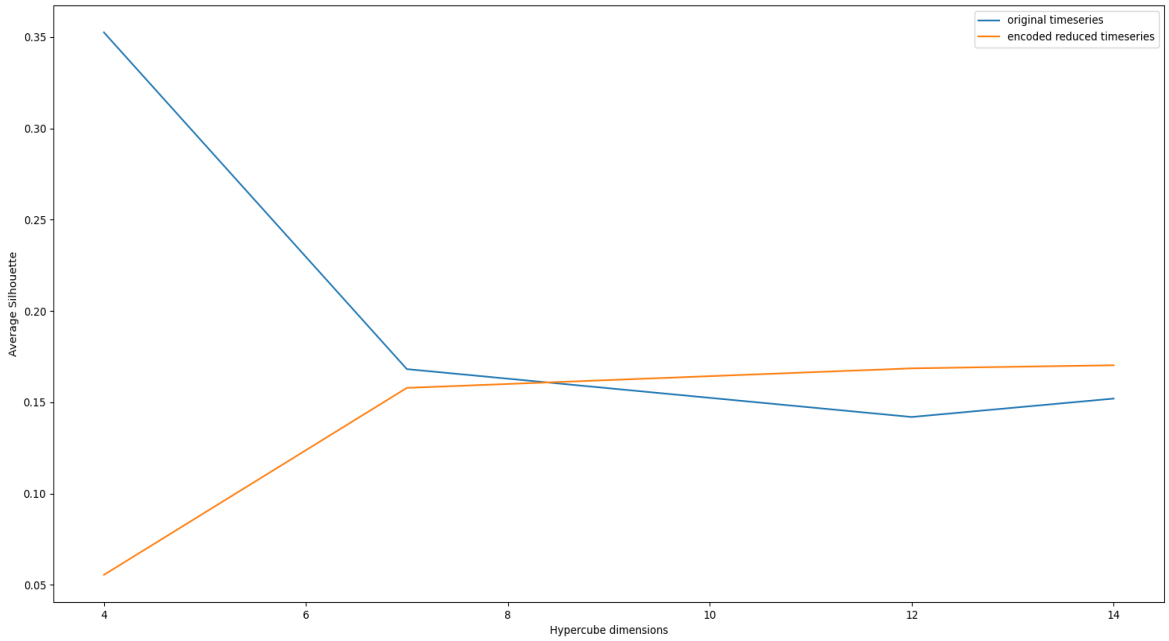
Silhouette results for original timeseries with 4 clusters using LSH Frechet

k/L	3	5	7
3	[0.884,0,0,-0.118,0.192]	[0.894,0,0,0.345,0.310]	[0,0,0,0.679,0.17]
4	[0.018,0,0,0.691,0.177]	[0.873,0,0,0.392,0.316]	[0.679,0,0,0,0.17]
5	[0.873,0,0,0.392,0.316]	[0.883,0,0,0,0.221]	[0.673,0,0,0,0.1684]

Silhouette average results for original/encoded_reduced timeseries with 4 clusters using LSH Mean Vector and k=5



Silhouette average results for original/encoded_reduced timeseries with 4 clusters using Hypercube Mean Vector



Εν τέλη είναι λογικό να έχουμε καλύτερο διαμοιρασμό των curves σε clusters αφού η μείωση των διαστάσεων βοηθά στην διατήρηση των περισσότερων σχετικών πληροφοριών στα δεδομένα που απαιτούνται για την εκμάθηση ακριβών, προγνωστικών μοντέλων.Ακόμη η μείωση αυτή αφαιρεί irrelevant features από το dataset, αφού η ύπαρξη τους μπορεί να μειώσει την ακρίβεια των μοντέλων και να κάνει train το μοντέλο βασισμένο σε irrelevant features.