Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα– Χειμερινό '23

3η Άσκηση - Readme

Α) **Τίτλος Προγράμματος** : Διανυσματική αναπαράσταση εικόνας σε χαμηλότερη διάσταση. Το *project* ξεκινά με την ανάγκη δημιουργίας ενός *autoencoder* για την αναπαράσταση των εικόνων σε μικρότερη διάσταση. Έπειτα, τα νέα σύνολα με εικόνες χρησιμοποιούνται ως $inputs$ σε κάποιους από τους αλγορίθμους των προηγούμενων παροδοτέων. Συγκρίνονται τα αποτελέσματα των αλγορίθμων στον νεο χώρο με τα αποτελέσματα στον αρχικό χώρο.
Β) Το παραδοτέο αποτελείται από 3 υποφακέλους, ένας για κάθε μια από τις 3 εργασίες του *project*. ο υποφάκελος που εξετάζει η εργασία αυτή είναι ο 3 (*ergasia*3). Στον φάκελο αυτό έχουμε συμπεριλάβει, πέραν των νέων αρχείων που απαιτεί η άσκηση, και τα αρχεία των πρώτων παραδοτέων. Αυτό γιατί αφενός τα χρησιμοπούμε όλα για την σύγκριση των αποτελεσμάτων αφετέρου σε πολλά από αυτά έχουν γίνει κάποιες αλλαγές, αφού στην εργασία αυτή ζητείται το μέσο κλάσμα προσέγγισης και όχι το μέγιστο όπως είχε υλοποιηθεί στην εργασία 2. Έχουν ακόμη προστεθεί κάποια αρχεία με *main* συναρτήσεις που με κατάλληλο τρόπο τρέχουν τους αλγόριθμους των 2 πρώτων εργασιών. Για να το πετύχουν αυτό πρέπει με κάποιο τρόπο να γίνει η αναγωγή από τον νέο χώρο στον αρχικό, γι΄ αυτό παίρνουν σαν όρισμα τόσο τα αρχεία με την αναπαράσταση των εικόνων στον νέο χώρο όσο και τα αρχεία με την αναπαράσταση των εικόνων στον νέο χώρο. Για να μην γίνουμε κουραστικοί, θα χωρίσουμε κάποια από τα αρχεία σε ομάδες και θα τα σχολιάσουμε συνολικά και όχι ένα ένα. Επίσης οι σχολιασμοί θα είναι σύντομοι και δεν θα αφορούν ιδιαίτερα το περιεχόμενο του κώδικα, αλλά την λειτουργία του. Άλλωστε θα προχωρήσουμε σε λεπτομερή ανάλυση των επιλογών και των αποτελεσμάτων πιο μετά. Σημειώνουμε επίσης ότι πολλά από τα αρχεία είναι *notebooks*, επιλέξαμε αυτόν τον τύπο για να μπορούν να παρουσιαστούν τα αποτελέσματα χωρίς να χρειαστεί να τρέξει κανείς τους κώδικες από την αρχή. Αυτό γιατί αφενός τα πειράματα παίρνουν αρκετό χρόνο αφετέρου για να τρέξουμε κάποια από τα πειράματα χρειάστηκε να προσθέσουμε κάποια μηνύματα στο *stdout* αλλά και να βγάλουμε κάποια *loops* τα οποία δεν ήταν μέρος των ζητουμένων και μετά τα αφαιρέσαμε. Αυτό σημαίνει ότι για να τρέξει κανείς τα πειράματα θα χρειαστεί να τροποποιήσει τα πηγαία αρχεία, λεπτομέρειες για το πως μπορεί να γίνει αυτό βρίσκονται στα σχόλια των πηγαίων αυτών αρχείων.

- *autoencoder_experiments.ipynb* : Το αρχείο στο οποίο γίνονται τα πειράματα για την εύρεση του καλύτερου δικτύου για τον *autoencoder*. Διαβάζει το αρχείο με τις εικόνες στον αρχικό χώρο, και μετά δημιουργεί ένα *optuna study* για να βελτιστοποιήσει τις υπερπαραμέτρους βάσει του *MSE loss* . Επιστρέφει την μικρότερη τιμή που βρήκε και τις αντίστοιχες υπερπαραμέτρους. Το αρχείο αυτό το τρέξαμε στο *kaggle* καθώς δεν είχαμε δικιάς μας *GPU* που να ήταν συμβατή με την *tensorflow* οπότε δεν μπορεί κανείς να το τρέξει άμεσα.

- *bottleneck_test* : Ένα σύνολο από αρχεία τύπου *notebook* που υπολογίζουν και προβάλουν το γράφημα του *MAF* ή της *Silhouette* καθώς και του χρόνου εκτέλεσης καθώς το *bottleneck size* μεταβάλλεται.

- *brute_force_reduced.cpp* : Αρχείο με μια *main* που παίρνει σαν ορίσματα από την γραμμή εντολών τα ονόματα του *dataset* και του *queryset* για την αρχική και την νέα διάσταση και υπολογίζει με εξαντλητική αναζήτηση τους κοντινότερους γείτονες στην αρχική διάσταση και μετά κάνει την αναγωγή στην αρχική διάσταση. Μεταγλώττιση με *make bruteReduced* εκτέλεση με ./bruteReduced -d <path to dataset> -q <path to queryset> -o <path to output file> -dr <path to reduced dataset> -qr <path to reduced queryset>

- *brute_force.cpp* : Αρχείο για την απλή εύρεση του κοντινότερου γείτονα στην αρχική διάσταση, χρησιμοποείται για την πιο εύκολη σύγκριση των αποτελεσμάτων. Μεταγλώττιση με *make brute* και εκτέλεση με ./bruteReduced -d <path to dataset> -q <path to queryset> -o <path to output file>

- *GetTrueDistances.cpp* : Αρχείο με μια συνάρτηση με το ίδιο όνομα η οποία παίρνει σαν όρισμα ένα *vector* από *ints* που αντιστοιχεί σε *indexes* των εικόνων και το *Query* σε *uint*8∗ αναπαράσταση και επιστρέφει έναν πίνακα από *doubles* που είναι οι αποστάσεις των εικόνων από τα *indexes*. Για να αναδειχθεί η χρησιμότητα αυτής της συνάρτησης πρέπει να εξετάσουμε τον τρόπο με τον οποίο αποθηκεύονται οι εικόνες. Αυτό

έχει αναλυθεί αρκετά στο *readme* και στα σχόλια του πρώτου παραδοτέου, αρκεί εδώ να πούμε πως μέσω των $interfaces$ των *ReadTrainData* και *ReadQueryData* μόνο ένα *dataset* και ένα *queryset* μπορεί να είναι φορτωμένο την κάθε στιγμή. Έτσι για τις συναρτήσεις στον νέο χώρο αρχικά φορτώνεται το *dataset* στο νέο χώρο, αποθηκεύονται τα *indexes* (κοινά για νέο και αρχικό χώρο), φορτώνεται το *dataset* στο νέο χώρο και καλείται η υπό συζήτηση συνάρτηση για να υπολογιστούν οι πραγματικές αποστάσεις.

- *Graph_Search_Reduced.cpp* : Αρχείο με *main* παρόμοιο με το αντίστοιχο αρχείο από το παραδοτέο της δεύτερης εργασίας μόνο που υπολογίζει του κοντινότερους γείτονες στον νέο χώρο. Μεταγλώττιση με *make graphReduced* και εκτέλεση με τον τρόπο που ορίζει η δεύτερη εργασία συν τα δύο ορίσματα -dr <path to reduced dataset> -qr <path to reduced queryset>

- *KMeansMainReduced.cpp* : Αρχείο με *main* παρόμοιο με το αντίστοιχο αρχείο από το παραδοτέο της πρώτης εργασίας μόνο που συσταδοποιεί στον νέο χώρο. Μεταγλώττιση με *make clusterReduced* και εκτέλεση με τον τρόπο που ορίζει η δεύτερη εργασία συν το όρισμα -dr <path to reduced dataset>

- *lsh_main_reduced.cpp* : Αρχείο με *main* παρόμοιο με το αντίστοιχο αρχείο από το παραδοτέο της δεύτερης εργασίας μόνο που υπολογίζει του κοντινότερους γείτονες στον νέο χώρο. Μεταγλώττιση με *make lshReduced* και εκτέλεση με τον τρόπο που ορίζει η δεύτερη εργασία συν τα δύο ορίσματα -dr <path to reduced dataset> -qr <path to reduced queryset>

- *optimize_* : Σύνολο *notebooks* που ξεκινούν με την συμβολοσειρά *optimize_* και βελτιστοποιούν τις υπερπαραμέτρους για τους αλγορίθμους *lsh* (στον αρχικό και στον νέο χώρο) *cube* (στον αρχικό χώρο) *gnn* (στον αρχικό και στον νέο χώρο). Επιστρέφουν τις βέλτιστες τιμές και αντίστοιχες υπερπαραμέτορυς για τον κάθε αλγόριθμο.

- *reduce.py* : Το αρχείο που ζητείται από την εκφώνηση.

Περιλαμβάνονται ακόμη τα αρχεία από τα προηγούμενα δύο παραδοτέα με μικροαλλαγές. Επίσης έχουμε συμπεριλάβει και τα αρχεία στον νέο χώρο σε περίπτωση που θέλει κανείς να τρέξει τα *notebooks* να μην χρειάζεται να τα δημιουργήσει εκ νέου.
Γ) **Μεταγλώττιση Προγραμμάτων**Για την μεταγλώττιση όλων των προγραμμάτων αρκεί η εντολή *make all*. Επειδή υπάρχει ένα θέμα με τις ονομασίες στο *cluster*, μάλλον θα χρειαστεί και η εντολή *make clusterReduced*.
Δ) **Εκτέλεση προγραμμάτων** Για την εκτέλεση των προγραμμάτων μπορούν να χρησιμοποιηθούν οι εντολές στην γραμμή εντολών όπως περιγράφηκαν παραπάνω και στις πρώτες δύο εργασίες. Ωστόσο υπάρχουν και αρχεία *notebooks* τα οποία μπορεί να τρέξει κανείς αν θέλει μέσω των *cells*. Τα αποτελέσματα μπορεί να διαφέρουν λόγω των τυχαιοκρατικών αλγορίθμων που χρησιμοποιούνται. Σημειώνουμε ότι για να μπορέσουμε να χρησιμοποιήσουμε το *optuna* αφαιρέσαμε το $do-while$ στους *LSH* και *Hypercube* και προσθέσαμε σε όλα τα αρχεία μια επιπλέον έξοδο στο *stdout* για να γίνει το *intercommunication* μεταξύ $c++$ και *python*.

Ε) Στοιχεία φοιτητών

- Απόστολος Κουκουβίνης, ΑΜ : 1115202000098

- Γιώργος Μητρόπουλος, ΑΜ : 1115202000128

Θα συνεχίσουμε τώρα με κάποια επεξηγηματικά σχόλια και πειράματα. Αρχικά όσον αφορά των *autoencoder* επιλέξαμε να βελτιστοποιήσουμε τις υπερπαραμέτρους μέσω του *optuna framework*. Οι υπερπαράμετροι προς βελτιστοποίηση είναι :

- *n_conv_layers* : πλήθος *convolutional layer*, οι τιμές είναι από 2 έως 5. Μερικά μόνο πειράματα αρκούν για να καταλάβει κανείς πως τα καλύτερα αποτελέσματα λαμβάνονται για 2-3 φίλτρα.

- *activation* : *activation function* οι επιλογές είναι *tanh, sigmoid, relu*

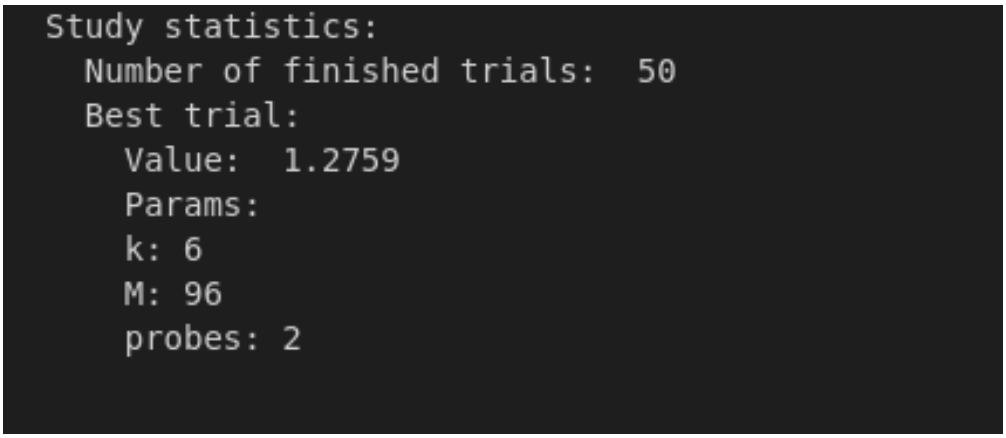
- *dropout rate* : τιμές 0-0.1
- $filters\_\{i\}$: Πλήθος φίλτρων στο $i-$οστο *convolutional layer*, οι καλύτερες τιμές εμφανίζοτνται για πλήθη κοντά στο 40, έτσι οι τιμές είναι 16-64
- $kernels\_\{i\}$: Μέγεθος του φίλτρου στο $i-$οστο *layer*, τιμές $3x3, 5x5$
- *bottleneck_size* : το *latent dimension*, στην περιοχή των συζητήσεων στο $e-class$ είχε γραφτεί πως πρέπει το *latent dimension* να είναι ˜ 30, δοκιμάζουμε μέχρι 40 αλλά στην πραγματικότητα όπως θα δούμε παρακάτω η βέλτιστη τιμή είναι για 15, τιμές από 10-40.
- *lr* : τιμές για το *LearningRate* από $1e-3$ έως $1e-1$
- *optimizer* : επιλογές είναι $adam, sgd, rmsprop, adamw$. Αν επιλεχθεί ο *optimizer sgd* τότε γίνεται *optimize* και στην τιμή της παραμέτρου *momentum*. Αν επιλεχθεί ο *optimizer rmsprop* τότε γίνεται *optimize* και στην τιμή της παραμέτρου *rho*. Αν επιλεχθεί ο *optimizer adamw* τότε γίνεται *optimize* και στην τιμή της παραμέτρου $weight_decay$.
- *batch_size* : τιμές 64-512
- *epochs* : τιμές 5-15.

Χρησιμοποιείται ακόμη *EarlyStopping* και *LearningRateScheduler* για την αποφυγή του $overfitting$. Για τα *pooling layers* έχουμε χρησιμοποιήσει μείωση της διάστασης κατά 2, δηλαδή δια 2. Το πρόβλημα που ανακύπτει είναι το εξής : με διαδοχικές διαρέσεις μεταβαίνουμε στις εξής διαστάσεις : 28-14-7-3. Ωστόσο, όταν κάνουμε το $upsampling$ επί 2 την διάσταση θα πάμε στις διαστάσεις 3-6-12-24. Η οποία δεν είναι η αρχική διάσταση. Για τον λόγο αυτό, επιλέξαμε αφενός να μην κατεβαίνουμε κάτω από την διάσταση 3, δηλαδή να μην χρησιμοποιείται *pooling* όταν η διάστααση είναι κάτω από 3, αφετέρου κατά το $upsampling$ από την διάσταση 3 να πηγαίνουμε στην διάσταση 7. Για να γίνει αυτό χρησιμοποιείται *padding* με την $ZeroPadding2D$ της $tensorflow$. Χρησιμοποιούνται ακόμη *BatchNormalization* και *Dropout* σε κάθε *convolutional layer*. Ακόμη χρησιμοποιείται ένα μόνο *Dense Layer* που πηγαίνει από το τελευταίο *convolutional* στο *bottleneck_size*. Το $flatten$ δεν γίνεται εδώ αλλά μόνο στο τελικό *encoder*. Αφού ολοκληρωθεί το *optuna study* για αυτές τις υπερπαραμέτρους και για 50 επαναλήψεις λαμβάνουμε της εξής καλύτερη προσπάθεια :

```
Study statistics:
  Number of finished trials:  50
  Best trial:
    Value:  0.0015369041357189417
    Params:
    bottleneck_size: 15
    n_conv_layers: 2
    activation: relu
    dropout_rate: 0.00044795018064423126
    filters_layer_0: 45
    kernel_size_layer_0: 5x5
    filters_layer_1: 46
    kernel_size_layer_1: 5x5
    optimizer: adam
    lr: 0.0141257629898787 94
    batch_size: 223
    epochs: 15
```

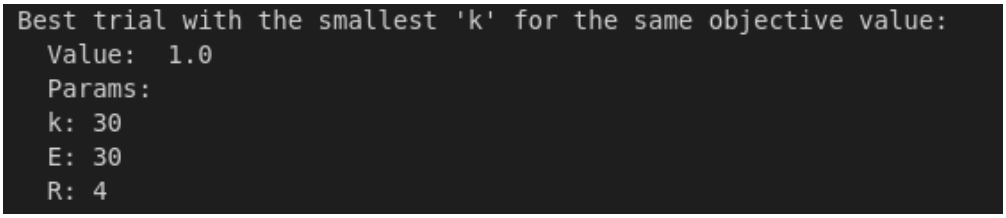
Ορίζουμε έτσι τον *encoder* με τις κατάλληλες παραμέτρους στο *reduce.py* και μπορούμε να δημιουργήσουμε τα *reduced datasets*. Όπως βλέπουμε, το *latent dimension* είναι 15. Το *latent dimension* είναι μάλλον η σημαντικότερη μεταβλητή αφού στην πραγματικότητα είναι ο χώρος στον οποίο μπορεί να αποθηκευτεί πληροφορία : διαισθητικά καταλαβαίνουμε πως όσο μεγαλύτερο το *latent dimension* τόσο πιο μεγάλη ακρίβεια αλλά και μεγαλύτερος χρόνο επεξεργασίας. Πρέπει να βρούμε μάλλον την χρυσή τομή η οποία πιθανότατα θα διαφέρει από αλγόριθμο σε αλγόριθμο. Θα πούμε περισσότερα γι' αυτό αργότερα. Προς το παρόν συνεχίζουμε με *optimizers* πάλι με *optuna studies* για το κάθε εκτελέσιμο αλγορίθμου. Για τα εν λόγω *optimizers* έχουμε χρησιμοποιήσει *notebooks* και για να μπορεί η *python* να λάβει το αποτέλεσμα, έχουμε τροποποιήσει τα $c++$ αρχεία για να τυπώνουν τα MAF με κατάλληλο $format$ στο *stdout*, το *optimize* γίνεται βάσει του MAF. Σημειώνουμε επίσης, ότι για λόγους χρόνου τα πειράματα έγιναν σε μικρότερο *dataset* (10.000 εικόνες) και μόνο 20 *queries*, ωστόσο πιστεύουμε πως είναι αντιπροσωπευτικά. Τέλος για λόγους επίδειξης και μόνο τυπώνουμε και το *importance* της κάθε υπερπαραμέτορυ στο τέλος του κάθε *notebook* Έτσι έχουμε :

- *Optimizer* για τον υπερκύβο στην αρχική διάσταση δοκιμάζουμε τιμές στο k μέσα στο 5-12, *probes* μέσα στο 1-2 και για το M στο 10-100 και παίρνουμε :



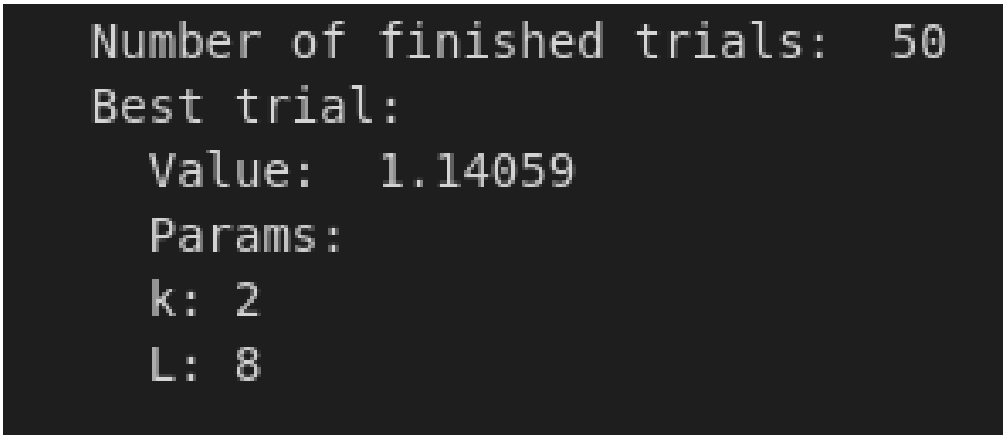
Με την πιο σημαντική μεταβλητή να είναι η k.

- *Optimizer* για τον gnn στην αρχική διάσταση δοκιμάζουμε τιμές στο k μέσα στο 30-100, R μέσα στο 1-4 και για το E στο 20-150 και παίρνουμε :



Με την πιο σημαντική μεταβλητή να είναι η E. Εδώ είχαμε πολλά σετ παραμέτρων με $MAF = 1$, έτσι επιλέγουμε αυτό με το μικρότερο K που είναι ο συντελεστής που επηρεάζει περισσότερο την ταχύτητα του αλγορίθμου.

- *Optimizer* για τον lsh στην αρχική διάσταση δοκιμάζουμε τιμές στο k μέσα στο 2-30, L μέσα στο 2-10 και παίρνουμε :



Με την πιο σημαντική μεταβλητή να είναι η L .

- *Optimizer* για τον gnn στην νέα διάσταση δοκιμάζουμε τιμές στο k μέσα στο 30-100, R μέσα στο 1-4 και για το E στο 20-150 και παίρνουμε :

```
Number of finished trials:  50
Best trial:
  Value:  1.37728
  Params:
  k: 45
  E: 87
  R: 1
```

Με την πιο σημαντική μεταβλητή να είναι η k με μικρή διαφορά από την E.

- *Optimizer* για τον *lsh* στην νέα διάσταση δοκιμάζουμε τιμές στο k μέσα στο 2-30, L μέσα στο 2-10 και παίρνουμε :

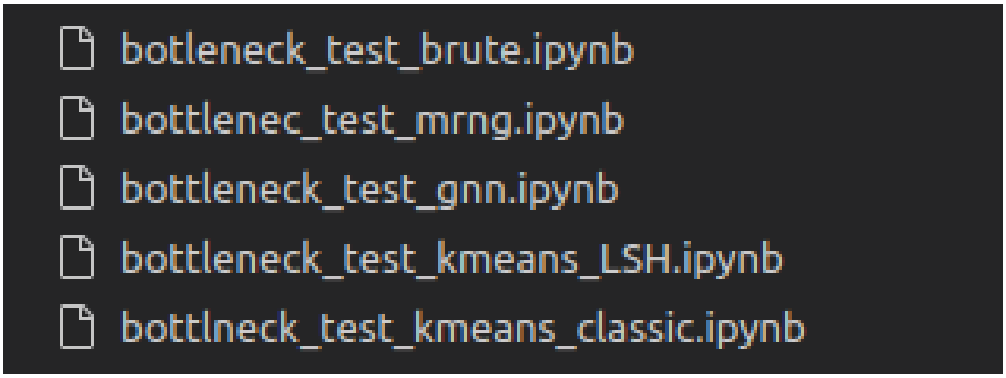
```
Study statistics:
  Number of finished trials:  50
  Best trial:
    Value:  1.51442
    Params:
    k: 30
    L: 10
```

Με την πιο σημαντική μεταβλητή να είναι η L με μικρή ωστόσο διαφορά.

Σημειώνουμε πως ενώ δεν ζητείται από την εκφώνηση LSH στον νέο χώρο εμείς από περιέργεια και μόνο τον δοκιμάζουμε. Αυτές λοιπόν είναι οι υπερπαράμετροι για τον κάθε αλγόριθμο που θα χρησιμοποιήσουμε στην συνέχεια. Σημειώνουμε πως για τον $MRNG$ είχαμε δει αποτελέσματα από το προηγούμενο παραδοτέο και ήταν κάπως έτσι :

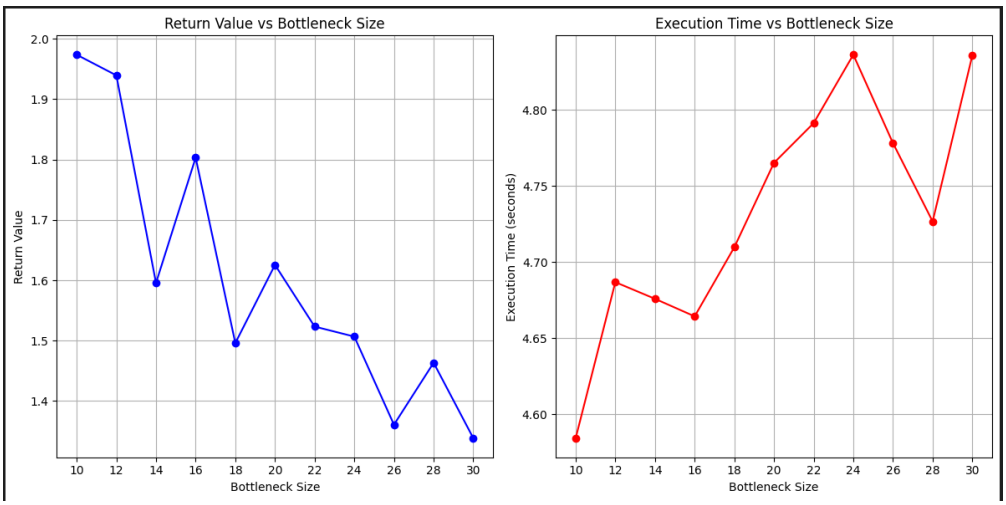
29	MRNG	-	-	-	-	-	-	20	1.39166	0.0001126
30	MRNG	-	-	-	-	-	-	50	1.24378	0.0004288
31	MRNG	-	-	-	-	-	-	100	1.00521	0.0015288
32	MRNG	-	-	-	-	-	-	400	1	0.0185454

Δηλαδή σχεδόν τέλειο για Λ 100 και σχετικά γρήγορο, αν θέλουμε κάτι εξαιρετικά γρήγορο τότε θα πηγαίναμε σε MAF κοντά στο 1.4 με Λ 20. Θα χρησιμοποιήσουμε $L = 150$ στον νέο χώρο και 100 στον αρχικό. Πριν συνεχίσουμε με σύγκριση των αποτελεσμάτων θα δούμε πως επηρεάζεται το MAF και το $Shilouette$ αλλά και ο συνολικός χρόνος εκτέλεσης στις αλλαγές του $latent\ dimension$. Για να γίνει αυτό, έχουμε δημιουργήσει αρχεία για διαστάσεις 10,12,14,...28,30 και θα εκτελέσουμε τα προγράμματα μας για αυτά τα αρχεία, αυτό γίνεται μέσω των $bottleneck\_test\ notebooks$:



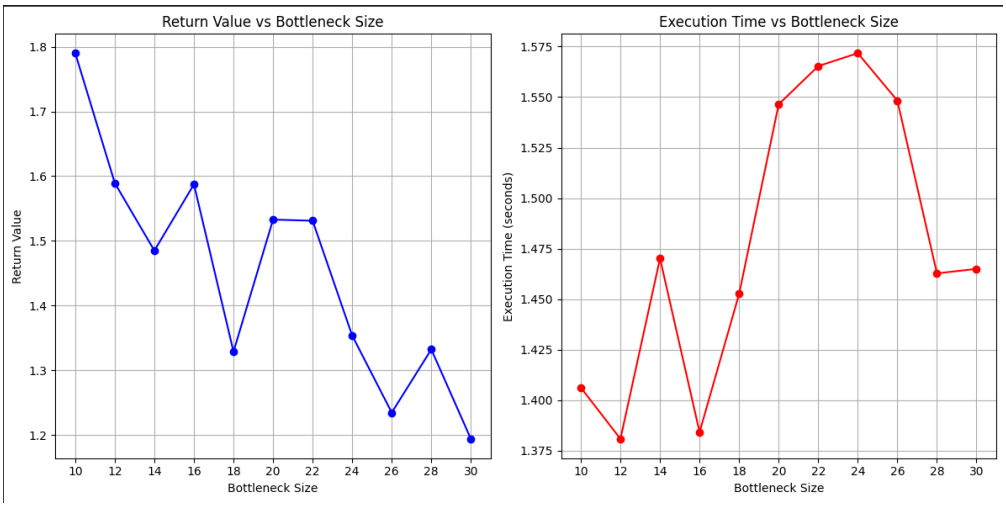
Έτσι έχουμε τα εξής αποτελέσματα:

- *BruteForce*



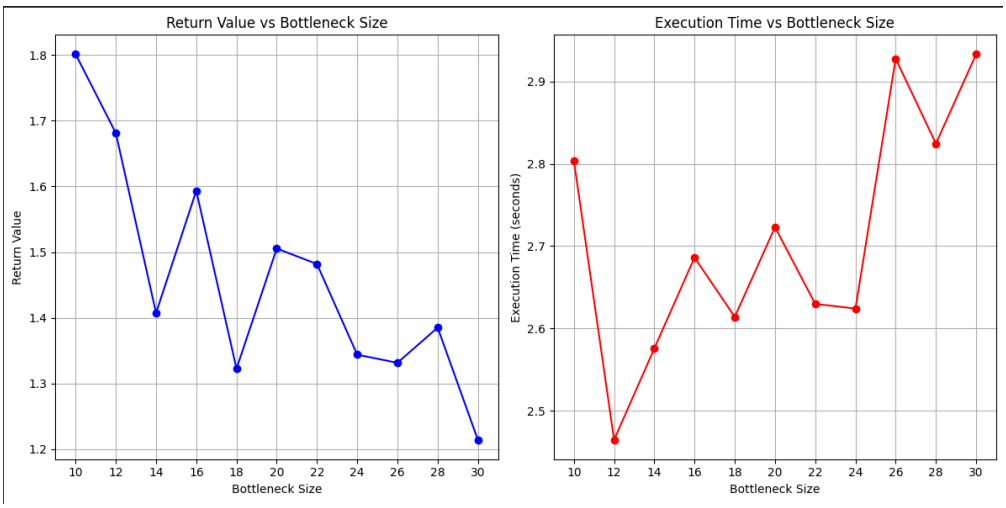
Παρατηρούμε δηλαδή ότι όσο αυξάνεται η διάσταση και προσεγγίζει το 30, τα αποτελέσματα γίνονται σημαντικά καλύτερα, ο χρόνος αυξάνεται αλλά η αύξηση δεν είναι ιδιαίτερα σημαντική, αρκεί να σημειώσουμε ότι είναι ο χρόνος για όλο το $dataset$. Αξίζει δηλαδή για το $brute$ να χρησιμοποιήσουμε το αρχείο με την διάσταση 30.

- *MRNG*



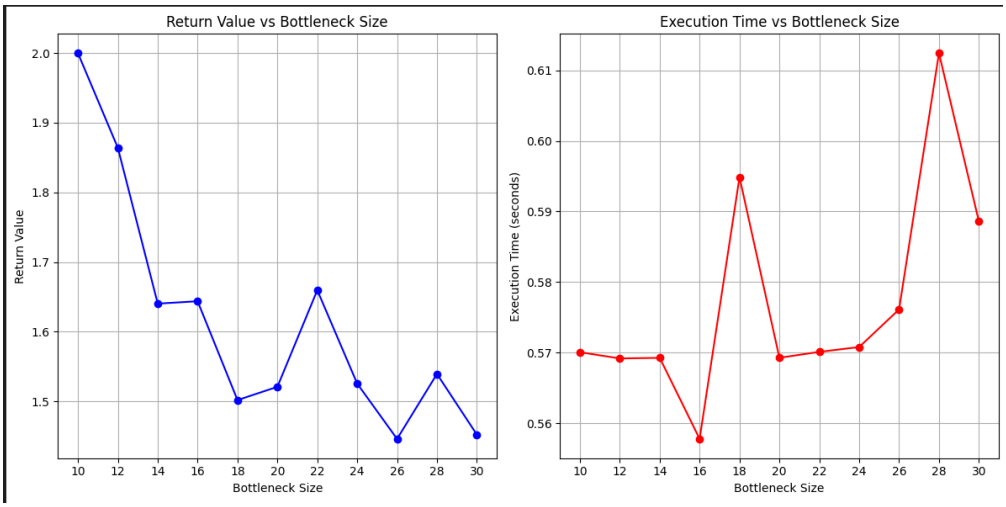
Παρατηρούμε δηλαδή ότι όσο αυξάνεται η διάσταση και προσεγγίζει το 30, τα αποτελέσματα γίνονται σημαντικά καλύτερα, ο χρόνος αυξάνεται αλλά εδώ έχουμε χρησιμοποιήσει μόνο τις 5000 εικόνες από το *dataset*, οπότε ίσως η αύξηση να είναι σημαντική. Θα το ελέγξουμε στο πλήρες *dataset* αργότερα.

- *GNN*



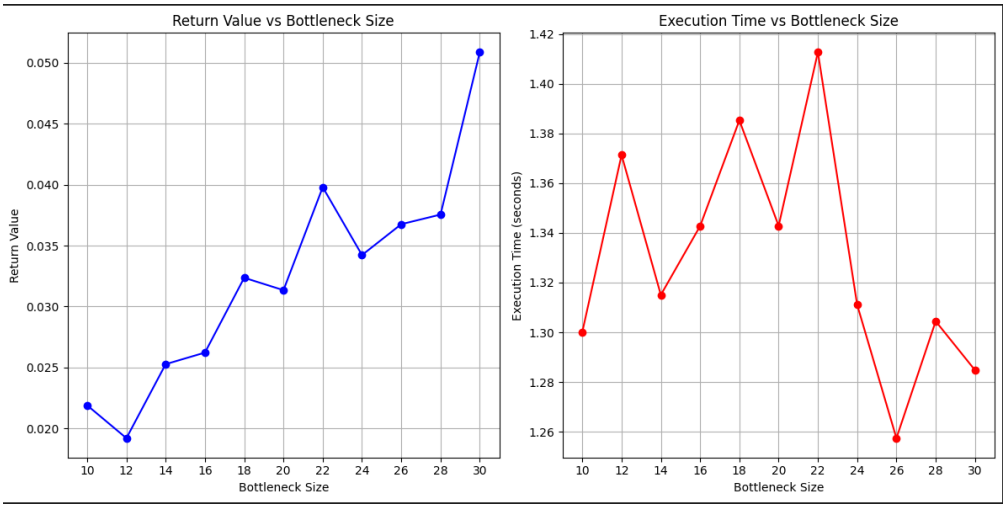
Παρατηρούμε δηλαδή ότι όσο αυξάνεται η διάσταση και προσεγγίζει το 30, τα αποτελέσματα γίνονται σημαντικά καλύτερα, ο χρόνος αυξάνεται αλλά εδώ έχουμε χρησιμοποιήσει μόνο τις 5000 εικόνες από το *dataset*, οπότε ίσως η αύξηση να είναι σημαντική. Θα το ελέγξουμε στο πλήρες *dataset* αργότερα.

- *LSH*



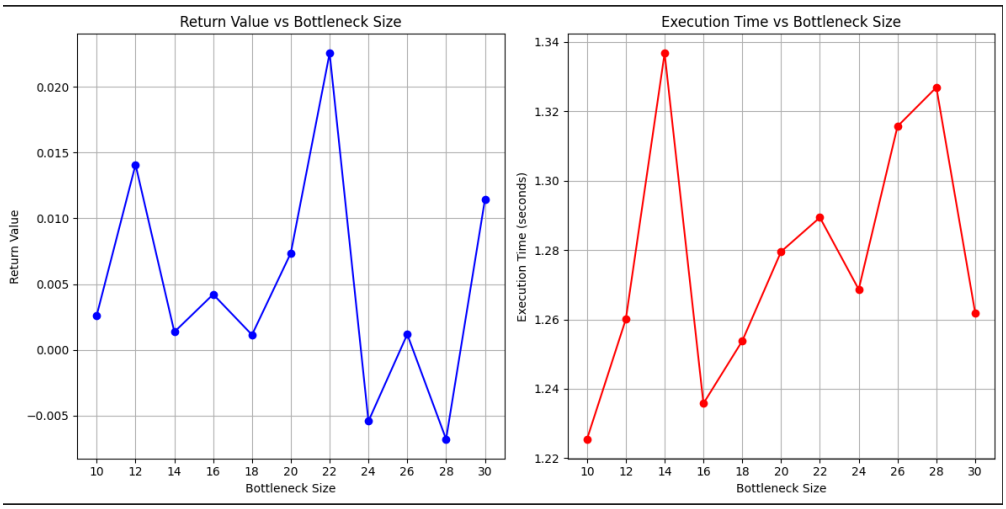
Παρατηρούμε δηλαδή ότι όσο αυξάνεται η διάσταση και προσεγγίζει το 30, τα αποτελέσματα γίνονται σημαντικά καλύτερα, ο χρόνος αυξάνεται αλλά εδώ έχουμε χρησιμοποιήσει μόνο τις 5000 εικόνες από το *dataset*, οπότε ίσως η αύξηση να είναι σημαντική. Θα το ελέγξουμε στο πλήρες *dataset* αργότερα.

- *KMeans – Lloyds*



Τώρα θέλουμε αύξηση της σιλουέτας. Παρατηρούμε ότι όσο αυξάνεται η διάσταση και προσεγγίζει το 30, τα αποτελέσματα γίνονται σημαντικά καλύτερα, ο χρόνος αυξάνεται αλλά εδώ έχουμε χρησιμοποιήσει μόνο τις 5000 εικόνες από το *dataset*, οπότε ίσως η αύξηση να είναι σημαντική. Θα το ελέγξουμε στο πλήρες *dataset* αργότερα.

- *KMeans – LSH*



Τώρα θέλουμε αύξηση της σιλουέτας. Εδώ παρατηρούμε ότι τα αποτελέσματα δεν είναι τόσο καλά με αρκετή απροσδιοριστία και σχεδόν καθόλου γραμμικότητα. Μάλλον δεν είναι μια καλή προσέγγιση.

Εν γένει, αυτό που παρατηρούμε είναι ότι σε διάσταση μικρότερη του 15 τα αποτελέσματα δεν είναι τόσο καλά. Από το 15 και μετά, τα αποτελέσματα γίνονται σημαντικά καλύτερα και συνήθως λαμβάνουν την καλύτερη τιμή τους κοντά στο 30. Σημειώνουμε επίσης ότι οι τιμές και οι χρόνοι εκτελέσεις μπορεί να μην είναι ακριβώς γραμμικοί, αυτό οφείλεται αφενός στις μικρές διαφοροποιήσεις που επιφέρει μια αύξηση της διάστασης κατά 2 αφετέρου στην τυχαιοκρατική φύση των περισσότερων αλγορίθμων. Οι χρόνοι επίσης επηρεάζονται από την χρήση της *CPU*. Σε κάθε περίπτωση, θα πειραματιστούμε για τα *dataset* διάστασης 15 και 30 για όλες τις εικόνες και θα συγκρίνουμε τα αποτελέσματα. Προκύπτει το εξής πινακάκι για την αναζήτηση πλησιέστερων γειτόνων :

Index	Method	LatentDimension	MAF	SecsPerQuery
1	BruteForce	-	1	0.0334
2	BruteForceReduced	15	1.74062	0.000071
3	BruteForceReduced	30	1.33852	0.00132
4	LSH	-	1.14	0.0136
5	LSHReduced	15	1.94	0.0061
6	LSHReduced	30	1.58	0.0051
7	Hypercube	-	1.74	0.22
8	GNNReduced	15	2	0.0012
9	GNNReduced	30	1.32	0.0011
10	MRNGReduced (L = 150)	15	1.98	0.0037
11	MRNGReduced (L=400)	15	1.89	0.021
12	MRNGReduced (L = 150)	30	1.35	0.0047
13	MRNGReduced (L=400)	30	1.38	0.027
15	GNN (R=4)	-	1	0.030
15	GNN (R=1)	-	1	0.00135
16	MRNG (L=100)	-	1.70	0.00089
17	MRNG (L=400)	-	1.21	0.019

Πίναχας 1: Trials Nearest Neighbors

Σημειώνεται ότι το *GNNReduced* έκανε μόλις 30 λεπτά για την κατασκευή του γράφου και την παραγωγή

των αποτελεσμάτων για την νέα διάσταση, τόσο για την 15 όσο και την 30. Για τον $MRNG$ η κατασκευή και εκτέλεση γίνεται τώρα πολύ γρήγορη μόλις σε 13 λεπτά χωρίς όμως η ταχύτητα να είναι τόσο καλή όσο ο GNN. Μπορούμε να δοκιμάσουμε να αυξήσουμε λίγο το L ώστε να έχουμε μεγαλύτερη ακρίβεια. Δοκιμάσαμε γι' αυτό και για $L = 400$ όπου ο συνολικός χρόνος γίνεται 19 λεπτά. Για την κατασκευή του GNN στην αρχική διάσταση και την εκτέλεση των *queries* χρειάστηκαν συνολικά 90 λεπτά που δεν είναι και απελπιστικά πολλά για τα αποτελέσματα που παίρνουμε με $MAF = 1$ αλλά ο χρόνος είναι κοντά στον χρόνο του *BruteForce*. Αυτό σίγουρα επηρεάζεται από τον αριθμό των τυχαίων επανεκκινήσεων που είναι 4, θα κάνουμε και μια δοκιμή με $R = 1$. Ο συνολικός χρόνος μειώθηκε σε 80 λεπτά και το MAF είναι πολύ λίγο πάνω από 1. Για τον $MRNG$ στον αρχικό χώρο η κατασκευή και η εκτέλεση παίρνει 80 λεπτά. Πριν προχωρήσουμε σε ανάλυση και συμπεράσματα βάσει των αποτελεσμάτων θα παρουσιάσουμε και τον πίνακα που αφορά την συσταδοποίηση. Χρησιμοποιούμε τους *Lloyds*, LSH στον νέο και στον αρχικό χώρο καθώς και τον *Hypercube* στον αρχικό χώρο. Για τον νέο χώρο θα δοκιμάσουμε τόσο για το αρχείο με *latent dimension* ίση με 15 όσο και για ίση με 30. Για τους LSH και *Hypercube* χρησιμοποιούμε τις παραμέτρους που βρέθηκαν παραπάνω. Έτσι έχουμε :

Index	Method	LatentDimension	Silhouette	Mean Objective Function	Clustering Time	Total Time
1	Lloyds	-	0.086	2566840	35.2375	10 min 42 sec
2	LSH	-	0.080	2598390	49.4046	10 min 49 sec
3	Hypercube	-	0.01	2966080	71.2511	11 min 10 sec
4	LloydsReduced	15	0.013	3056480	0.32	9 min 14 sec
5	LloydsReduced	30	0.045	2809560	0.3	9 min 14 sec
6	LSHReduced	15	-0.014	3352740	90.3275	11 min 5 sec

Πίνακας 2: Trials Clustering

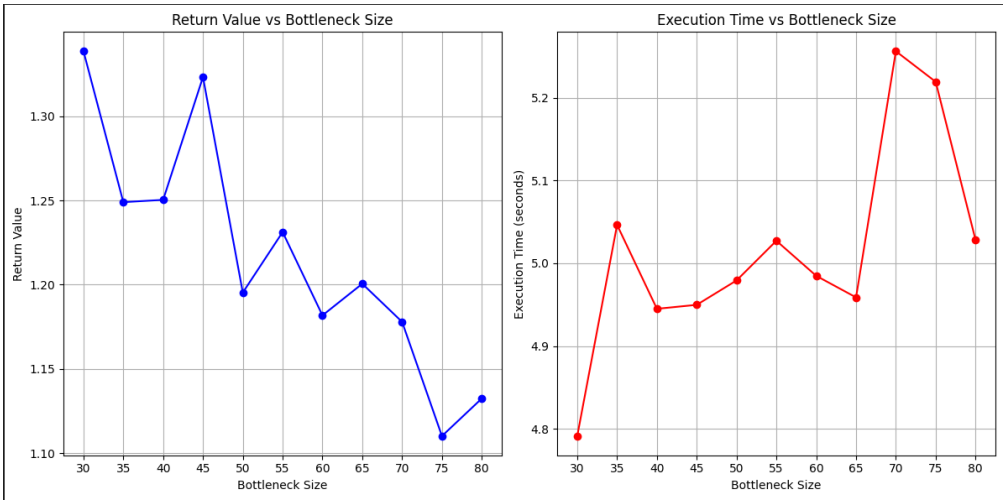
Συγκρίνουμε τους προσεγγιστικούς αλγόριθμους για *NearestNeighbor* στον νέο χώρο :

- *Brute Force :* Συγκρίνοντας τον *BruteForce* στην αρχική και στην νέα διάσταση παρατηρούμε ότι στην νέα διάσταση ίση με 15 είναι πάνω από 400 φορές ταχύτερος με MAF που θα μπορούσε να θεωρηθεί αποδεκτό, κοντά στο 1.74 ενώ στην νέα διάσταση το MAF γίνεται ιδιαίτερα ανταγωνιστικό σε σχέση με άλλους αλγόριθμους και ο χρόνος παραμένει γρήγορος περίπου 25 φορές πιο γρήγορος από τον *BruteForce* στην αρχική διάσταση. Σε σχέση με τον LSH στην αρχική διάσταση ο *BruteForce* είναι σαφώς ταχύτερος και στην διάσταση 15 όσο και στην διάσταση 30, αλλά δεν είναι τόσο καλό το MAF. Όσον αφορά υπερτερεί κατά κράτος σε ταχύτητα και ενώ το MAF είναι ίδιο για την διάσταση 15 στην διάσταση 30 το MAF γίνεται σημαντικά καλύτερο. Σε σχέση με τον GNN στην διάσταση 15 είναι περίπου 19 φορές ταχύτερος αλλά σε καμία περίπτωση δεν πλησιάζει το MAF του που είναι ίσο με 1 ενώ στην διάσταση 30 έχουν την ίδια ταχύτητα περίπου με το MAF να πλησιάζει σιγά σιγά την μονάδα. Σε σχέση με τον $MRNG$ για $L = 100$ στην διάσταση 15 είναι 12 φορές ταχύτερος ο *Brute Force* με λίγο χειρότερα αποτελέσματα στο MAF ενώ στην περίπτωση της διάσταση 30 ο *Brute Force* είναι λίγο πιο αργός αλλά το MAF είναι σημαντικά καλύτερο. Για τον $MRNG$ με $L = 100$ το MAF στην διάσταση 15 το MAF απέχει πολύ από αυτό το $MRNG$ αλλά η ταχύτητα είναι σαφώς καλύτερη ενώ στην διάσταση 30 τα MAF δεν απέχουν πολύ μεταξύ τους με τον *BruteForce* να παραμένει περίπου 14 φορές ταχύτερος.
- *LSH :* Για τον LSH στον νέο χώρο παρατηρούμε ότι θα επιλέγαμε σίγουρα την διάσταση 30, αλλά και πάλι σε καμία περίπτωση δεν θα έφτανε τον απλό *BruteForce*, τελικά θα τον απορρίπταμε με συνπτικές διαδικασίες.
- *GNN :* Για τον GNN στον νέο χώρο παρατηρούμε αρχικά ότι οι χρόνοι είναι περίπου οι ίδιοι και στις δύο διαστάσεις σε σχέση με τους χρόνους του GNN στον αρχικό χώρο. Τα MAF είναι αρκετά χειρότερα από το τέλειο MAF του GNN στην αρχική διάσταση. Ωστόσο, οι ταχύτητα κατασκευής του γράφου είναι περίπου 8 φορές καλύτερη. Και πάλι όμως μάλλον θα επιλέγαμε τον *BruteForce*

στην νέα διάσταση καθώς έχουμε περίπου ίδια MAF και χρόνους χωρίς να απαιτείται η κατασκευή του γράφου.

- *MRNG :* Πάλι ότι ισχύει και για τον GNN ισχύει και για τον $MRNG$ ίδιοι χρόνοι και χειρότερα MAF με σημαντικά μικρότερο χρόνο κατασκευής. Και πάλι ο *BruteForce* υπερτερεί.

Τελικά μπορούμε να πούμε το εξής : αν θέλαμε πολύ γρήγορη αναζήτηση και ένα σχετικό μεγάλο MAF δεν θα μας πείραζε, που παραμένει μικρότερο του 2, τότε θα επιλέγαμε τον *BruteForce* στην διάσταση 15. Αν θέλαμε κάτι πιο ακριβές θα μπορούσαμε να χρησιμοποιήσουμε τον *BruteForce* στην διάσταση 30 με MAF κοντά στο 1.33. Αν ζητούσαμε ακόμη μεγαλύτερη ακρίβεια, κοντά στο 1.14 χωρίς να κατασκευάσουμε γράφους, θα χρησιμοποιούσαμε LSH στον αρχικό χώρο. Αν θέλαμε σχεδόν τέλεια αναζήτηση, θα μπορούσαμε να χρησιμοποιούσαμε τον GNN που θα έδινε τέλεια αποτελέσματα παραμένοντας 24 φορές πιο γρήγορος από τον *BruteForce* στον αρχικό χώρο. Αυτό που έχει ενδιαφέρον είναι να εξετάσουμε αν περαιτέρω αύξηση του *latent dimension* θα οδηγούσε σε ακόμη καλύτερα αποτελέσματα. Δοκιμάζοντας για κάποιες ακόμη τιμές προκύπτει το εξής πινακάκι :



Βλέπουμε ότι υπάρχει μείωση σημαντική. Αν τρέξουμε τον *BruteForce* για την διάσταση 75 παίρνουμε τα εξής αποτελέσματα : μέσος χρόνος ανά $Query$ $0.00303724seconds$ και $MAF = 1.1102$ οπότε γίνεται καλύτερος από τον LSH τόσο σε χρόνο όσο και MAF. Καταλήγουμε ότι θα επιλέγαμε τον *BruteForce* στην νέα διάσταση αν θέλαμε κάτι άμεσο χωρίς επιπλέον δομές και θα προσαρμόζαμε κατάλληλα την διάσταση του νέου χώρου ανάλογα με το MAF που θέλαμε. Αν θέλαμε κάτι σχετικά γρήγορο και με τέλειο MAF και αντέχαμε να πληρώσουμε το κόστος για την κατασκευή επιπλέον δομών θα επιλέγαμε τον GNN.

Συγκρίνοντας το *Clustering* αρχικά παρατηρούμε ότι σε κάθε περίπτωση στον αρχικό χώρο το *Clustering* με *Lloyds* είναι η καλύτερη επιλογή, αφού είναι ταχύτερο και με καλύτερη σιλουέτα και συνάρτηση στόχου. Αυτό συμβαίνει γιατί επιτυγχάνεται γρήγορη σύγκλιση. Το ίδιο συμβαίνει και στον νέο χώρο με τα αποτελέσματα του LSH να είναι πολύ κακά. Οπότε η σύγκριση μπορεί να γίνει μεταξύ *Clustering* στον νέο χώρο και *Clustering* στον παλιό χώρο. Το *clustering* στον νέο χώρο είναι σχεδόν στιγμιαίο και παίρνει μόνο 0.3 *seconds* με την σιλουέτα να είναι περίπου δύο φορές χειρότερη. θα επιλέγαμε λοιπόν το *Clustering* στον νέο χώρο αν θέλαμε κάτι πολύ γρήγορο και δεν μας πείραζε η σιλουέτα να γίνει λίγο χειρότερη. Πρέπει να σημειώσουμε εδώ ότι για την αντικειμενική συνάρτηση χρησιμοποιούμε εδώ τον μέσο όρο ώστε να αποφύγουμε το $overflow$, αν θέλαμε την πραγματική τιμή θα πολλαπλασιάζαμε τον μέσο όρο με το πλήθος των εικόνων, δηλαδή 60000. Είναι σαφές ότι όταν προσπαθούμε να βελτιστοποιήσουμε την σιλουέτα, την αντικειμενική συνάρτηση ή τον μέσο όρο της αντικειμενικής συνάρτησης, όλα τα παραπάνω επηρεάζονται θετικά αφού όλα στο κάτω κάτω έχουν να κάνουν με τις αποστάσεις των σημείων από τα κέντρα τους.

Τέλος, αν δοκιμάσουμε για νέα διάσταση ίση με 75, τότε η σιλουέτα είναι κοντά στο 0.7, ακόμη πιο κοντά στο τέλειο δηλαδή, και ο χρόνος γίνεται 1.66932. Δηλαδή ακόμη είναι περίπου 30 φορές πιο γρήγορη συσταδοποίηση με πολύ καλά αποτελέσματα. Τελικά θα επιλέγαμε συσταδοποίηση με *Lloyds* στον νέο χώρο αν θέλαμε πιο γρήγορα αποτελέσματα και θα προσαρμόζαμε την διάσταση του νέου χώρου ανάλογα με την ανοχή μας στο σφάλμα. Πρέπει επίσης να σημειώσουμε εδώ, ότι τα παραδείγματα σε διάσταση μεγαλύτερη από 30 δεν ήτανε πολλά καθώς από αυτά που καταλάβαμε στο *eclass* δεν ζητείται κάτι τέτοιο. Επιπλέον βγάζει νόημα καθώς μετά καθυστερούν σημαντικά οι αλγόριθμοι.

Συνοψίζοντας, η κωδικοποίηση στον νέο χώρο είναι μια πολύ ενδιαφέρουσα τεχνική που είναι ιδιαίτερα ανταγωνιστική σε σχέση με τους αλγορίθμους των πρώτων εργασιών. Μάλλον δεν χρειάζεται συνδυασμός τους παρά μόνο το *BruteForce* για να έχουμε καλά αποτελέσματα.