Δεύτερη Σειρά Ασκήσεων – Εξόρυξη Δεδομένων

Όνομα: Χρύσα Τεριζή
ΑΜ: 2553
Ημερομηνία: 30/4/2017
Free passes: Δεν χρησιμοποίησα κανένα

Ερώτηση 1(Singular Value Decomposition)

1. Έστω, $M = U\Sigma V^T$ οπότε για το M^T ισχύει ότι, $M^T = (U\Sigma V^T)^T = V\Sigma U^T$. Για το M^TM ισχύουν τα ακόλουθα,
   $M^TM = (V\Sigma U^T)(U\Sigma V^T) = V\Sigma\Sigma V^T = V\Sigma^2V^T$ → είναι της μορφής $Q\Lambda Q^T$, άρα είναι τετράγωνος, συμμετρικός και πραγματικός σύμφωνα με την εκφώνηση.
   $MM^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma\Sigma U^T = U\Sigma^2U^T$ → είναι της μορφής $Q\Lambda Q^T$, άρα είναι τετράγωνος, συμμετρικός και πραγματικός σύμφωνα με την εκφώνηση.
2. Ξέρουμε ότι αν ένας πίνακας L είναι ορθογώνιος ισχύει ότι,
   $||Lx||_2^2 = = (Lx)^T(Lx) = x^TL^TLx = x^Tx = ||x||_2^2 \Rightarrow ||Lx||_2 = ||x||_2$

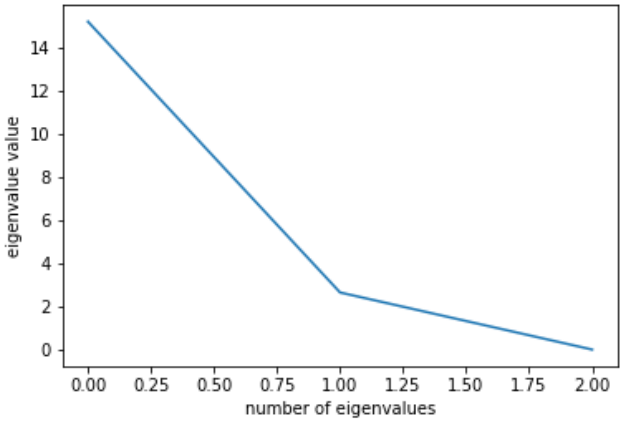
   Οι πίνακες M^TM και MM^T είναι ορθογώνιοι , ο M^TM έχει διάσταση (d, d) και ο MM^T έχει διάσταση (n, n). Οπότε θα ισχύει ότι,

   $||M^TMx||_2 / ||x||_2 = ||V\Sigma^2V^Tx||_2 / ||x||_2 = ||\Sigma^2V^Tx||_2 / ||x||_2 = ||\Sigma^2V^Tx||_2 / ||V^Tx||_2 =$
   $||\Sigma^2y||_2 / ||y||_2$ όπου $y = V^tx$.

   $||MM^Tx||_2 / ||x||_2 = ||U\Sigma^2U^Tx||_2 / ||x||_2 = ||\Sigma^2U^Tx||_2 / ||x||_2 = ||\Sigma^2U^Tx||_2 / ||U^Tx||_2 =$
   $||\Sigma^2y||_2 / ||y||_2$ όπου $y = U^tx$.

   Άρα, βρίσκουμε ότι, $||M^TMx||_2 / ||x||_2 = ||MM^Tx||_2 / ||x||_2 = ||\Sigma^2y||_2 / ||y||_2$.
3. Ισχύει ότι,
   $M^TM = (V\Sigma U^T)(U\Sigma V^T) = V\Sigma\Sigma V^T = V\Sigma^2V^T$.
4. Για τον συγκεκριμένο πίνακα που μας δίνεται βρίσκω ότι τα singular values στον Σ είναι τα ακόλουθα:
   [1.51986842e+01 2.64575131e+00 5.74916580e-16]

   Το rank του πίνακα αυτού είναι 2. Το μέγεθος του Σ είναι 3x3 άρα υπάρχει θόρυβος. Η τιμή 5.74916580e-16 είναι πάρα πολύ μικρή σε σχέση με τις υπόλοιπες οπότε είναι θόρυβος. Από την γραφική παράσταση βλέπουμε ότι ο πίνακας μας μπορεί να αναπαρασταθεί και με μία διάσταση αφού εκεί βλέπουμε να υπάρχει το λεγόμενο “γόνατο”. Αλλά επειδή υπάρχει θόρυβος δεν ξέρουμε αν όντως μπορεί να αναπαρασταθεί με μία διάσταση.



Ερώτηση 2(Min-Hashing & LSH)

Αρχικά, διαβάζω το αρχείο "***twitter_dataset.txt***" και βρίσκω για κάθε tweet το όνομα του χρήστη που το έχει κάνει. Με την συνάρτηση ***getTagsHandles(x)*** βρίσκω για κάθε tweet τα hashtags και handles τα οποία υπάρχουν στο συγκεκριμένο tweet. Μέσε σε ένα λεξικό "***dictionaryForHashtagsHandles***" κρατάω σαν κλειδί το όνομα του χρήστη και σαν τιμή μία λίστα με τα hashtags/handles που έχει χρησιμοποιήσει συνολικά. Βρίσκω ότι μέχρι στιγμής υπάρχουν:

Users: 305245
Hashtags & handles: 648150

Κάνω το πρώτο κλάδεμα, κρατώντας μόνο τους χρήστες με > 20 hashtags/handles και τα hashtags/handles που έχουν χρησιμοποιηθεί από > 20 διαφορετικούς χρήστες. Βρίσκω ότι,

Users: 17658
Hashtags & handles: 26661

Ακολουθεί το επαναληπτικό κλάδεμα, όπου για κάθε χρήστη βρίσκω την καινούρια του λίστα με τα hashtags/handles τα οποία hashtags/handles είναι πιο συχνά(> 20). Έπειτα, ξανά βρίσκω από αυτούς τους χρήστες ποιοι έχουν hashtags/handles > 20. Βρίσκω το καινούριο πλήθος από τα hashtags/handles και αφαιρώ αυτά που έχουν πλήθος < 20. Κάνω αυτήν την διαδικασία επαναληπτικά έως ότου δεν αλλάξει το σύνολο από hashtags/handles και οι χρήστες δεν μειώνονται. Οπότε σαν τελικό πλήθος από χρήστες και hashtags/handles βρίσκω ότι είναι,

Users: 5360
Hashtags & handles: 2684

Κατόπιν, μετατρέπω τα hashtags/handles σε 32-bits ακεραίους αριθμούς και θα έχουν την ακόλουθη μορφή,

@nissanusa → 1697985572
#heismanhouse → 535282446
@vine → 1361408837
@brithume → 3005783194
@donaldjtrumpjr → 2848451318

Φτιάχνω τον πίνακα "***tMinHashTable***" ο οποίος περιέχει 0 και 1. Στον κατακόρυφο άξονα υπάρχουν τα hashtags/handles και στον οριζόντιο άξονα υπάρχουν οι χρήστες. Κρατάω για κάθε χρήστη μία λίστα που περιέχει μόνο τους αριθμούς των γραμμών όπου περιέχουν 1 για να γίνει πιο γρήγορα η σύγκριση για να βρω την *jaccard ομοιότητα* τους. Έχω φτιάξει την συνάρτηση " ***jaccardSimilarity(x, y)***" η οποία υπολογίζει την ομοιότητα για 2 χρήστες παίρνοντας σαν όρισμα τις λίστες με τους αριθμούς των γραμμών(hashtags/handles) για τους χρήστες που θέλει να συγκρίνει. Βρίσκω την πραγματική ομοιότητα ***όλων των ζευγαριών(14362120)*** ανάμεσα στους χρήστες. Έπειτα, βρίσκω αυτούς οι οποίοι έχουν ομοιότητα > 0.80. Είναι **23** τα ζευγάρια αυτά και είναι τα ακόλουθα:

(RealAlexJones,infowars): 0.8372093023255814
(Able_49,Doug_39): 0.9692307692307692
(Able_49,Earl_59): 0.9545454545454546
(Able_49,Chad_59): 0.9541984732824428
(Able_49,ggeett37aaa): 0.9402985074626866
(Able_49,dansch2002): 0.8939393939393939
(Able_49,Bart_39): 0.9847328244274809
(Doug_39,Earl_59): 0.984375
(Doug_39,Chad_59): 0.984251968503937
(Doug_39,ggeett37aaa): 0.9541984732824428

(Doug_39,dansch2002): 0.921875
(Doug_39,Bart_39): 0.9692307692307692
(Earl_59,Chad_59): 0.9689922480620154
(Earl_59,ggeett37aaa): 0.9398496240601504
(Earl_59,dansch2002): 0.9076923076923077
(Earl_59,Bart_39): 0.9545454545454546
(Chad_59,ggeett37aaa): 0.9393939393939394
(Chad_59,dansch2002): 0.9069767441860465
(Chad_59,Bart_39): 0.9541984732824428
(ggeett37aaa,dansch2002): 0.8796992481203008
(ggeett37aaa,Bart_39): 0.9259259259259259
(dansch2002,Bart_39): 0.8939393939393939
(Forprinciple16,goheels_70): 0.9714285714285714

Τώρα, είναι η σειρά για να βρίσκω τις ομοιότητες των ζευγαριών ανάλογα με τις hash function signatures. Για κάθε hashtag/handle έχω μία λίστα με τον αριθμό του χρήστη ο οποίος το έχει χρησιμοποιήσει για να υπολογιστή πιο γρήγορα η προσεγγιστική jaccard ομοιότητα ανάμεσα στους χρήστες. Με την συνάρτηση "***findHashSignatures()***" βρίσκω τις συναρτήσεις. Με την μεταβλητή "***howManyHashFunction***" δηλώνω πόσες συναρτήσεις θα βρω κάθε φορά. Για να τρέξω τις περιπτώσεις των 50, 100, 200 συναρτήσεων θα πρέπει να αλλάξει κάποιος αυτήν την μεταβλητή στον αριθμό που θέλει και να τρέξει ένα-ένα τα κελιά που ακολουθούν. Για παράδειγμα, η μορφή μίας συνάρτησης είναι η ακόλουθη,

Hash function: **3423077312X + 1992128029 mod 4294967311**

Εκτελώ τον αλγόριθμο για να βρω τις υπογραφές για κάθε συνάρτηση(όπως είναι γραμμένος και στις διαφάνειες στη σελίδα του μαθήματος). Η συνάρτηση "***createTableForHashes(itera)***" αρχικοποιεί τον πίνακα με ***άπειρο***(***2^50***). Μέσα στην λίστα "***listaMeOlesTisStiles***" κρατάω για κάθε χρήστη μία λίστα με τα αποτελέσματα για όλες τις υπογραφές για την κάθε συνάρτηση. Η συνάρτηση "***findJaccardSimilarityForHashFunction(x, y, div)***" υπολογίζει την προσεγγιστική jaccard ομοιότητα μεταξύ 2 χρηστών. Έπειτα, για όλους τους συνδυασμούς των χρηστών βρίσκω την παραπάνω ομοιότητα. Τα αποτελέσματα για να ζευγάρια όπου έχουν ομοιότητα > 0.8 είναι τα ακόλουθα:

Για 10 συναρτήσεις(pairs: 22):

(Able_49,Doug_39): 1.0
(Able_49,Earl_59): 1.0
(Able_49,Chad_59): 0.9
(Able_49,ggeett37aaa): 1.0
(Able_49,dansch2002): 1.0
(Able_49,Bart_39): 1.0
(Doug_39,Earl_59): 1.0
(Doug_39,Chad_59): 0.9
(Doug_39,ggeett37aaa): 1.0
(Doug_39,dansch2002): 1.0
(Doug_39,Bart_39): 1.0
(Earl_59,Chad_59): 0.9
(Earl_59,ggeett37aaa): 1.0
(Earl_59,dansch2002): 1.0
(Earl_59,Bart_39): 1.0
(Chad_59,ggeett37aaa): 0.9
(Chad_59,dansch2002): 0.9
(Chad_59,Bart_39): 0.9
(ggeett37aaa,dansch2002): 1.0
(ggeett37aaa,Bart_39): 1.0
(dansch2002,Bart_39): 1.0
(Forprinciple16,goheels_70): 1.0

Για 50 συναρτήσεις(pairs: 23):

(RealAlexJones,infowars): 0.84
(Able_49,Doug_39): 0.96
(Able_49,Earl_59): 0.96
(Able_49,Chad_59): 0.88
(Able_49,ggeett37aaa): 0.92
(Able_49,dansch2002): 0.88
(Able_49,Bart_39): 1.0
(Doug_39,Earl_59)': 1.0
(Doug_39,Chad_59): 0.92
(Doug_39,ggeett37aaa): 0.96
(Doug_39,dansch2002): 0.92
(Doug_39,Bart_39): 0.96

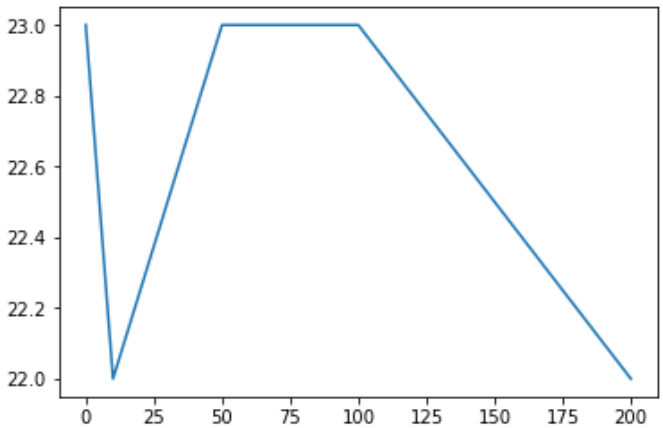
(Earl_59,Chad_59): 0.92
(Earl_59,ggeett37aaa): 0.96
(Earl_59,dansch2002): 0.92
(Earl_59,Bart_39): 0.96
(Chad_59,ggeett37aaa): 0.88
(Chad_59,dansch2002): 0.84
(Chad_59,Bart_39): 0.88
(ggeett37aaa,dansch2002): 0.88
(ggeett37aaa,Bart_39): 0.92
(dansch2002,Bart_39): 0.88
(Forprinciple16,goheels_70): 1.0

Για 100 συναρτήσεις(pairs: 23):

(RealAlexJones,infowars): 0.92
(Able_49,Doug_39): 0.97
(Able_49,Earl_59): 0.95
(Able_49,Chad_59): 0.95
(Able_49,ggeett37aaa): 0.91
(Able_49,dansch2002): 0.91
(Able_49,Bart_39): 0.98
(Doug_39,Earl_59): 0.98
(Doug_39,Chad_59): 0.98
(Doug_39,ggeett37aaa): 0.94
(Doug_39,dansch2002): 0.94
(Doug_39,Bart_39): 0.95
(Earl_59,Chad_59): 0.96
(Earl_59,ggeett37aaa): 0.93
(Earl_59,dansch2002): 0.92
(Earl_59,Bart_39): 0.93
(Chad_59,ggeett37aaa): 0.93
(Chad_59,dansch2002): 0.92
(Chad_59,Bart_39): 0.93
(ggeett37aaa,dansch2002): 0.88
(ggeett37aaa,Bart_39): 0.89
(dansch2002,Bart_39): 0.89
(Forprinciple16,goheels_70): 0.97

Για 200 συναρτήσεις(pairs: 22):

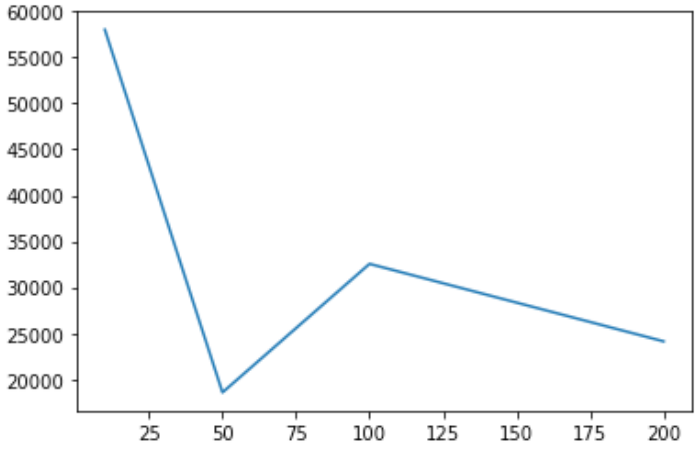
(Able_49,Doug_39): 0.945
(Able_49,Earl_59): 0.925
(Able_49,Chad_59): 0.93
(Able_49,ggeett37aaa): 0.92
(Able_49,dansch2002): 0.85
(Able_49,Bart_39): 0.975
(Doug_39,Earl_59): 0.98
(Doug_39,Chad_59): 0.985
(Doug_39,ggeett37aaa): 0.935
(Doug_39,dansch2002): 0.9
(Doug_39,Bart_39): 0.96
(Earl_59,Chad_59): 0.965
(Earl_59,ggeett37aaa): 0.915
(Earl_59,dansch2002): 0.88
(Earl_59,Bart_39): 0.94
(Chad_59,ggeett37aaa): 0.92
(Chad_59,dansch2002): 0.885
(Chad_59,Bart_39): 0.945
(ggeett37aaa,dansch2002): 0.84
(ggeett37aaa,Bart_39): 0.895
(dansch2002,Bart_39): 0.86
(Forprinciple16,goheels_70): 0.97



Συμπεράσματα από το γράφημα: Ότι το πλήθος των ζευγαριών τα οποία έχουν ομοιότητα > 0.8 για το διαφορετικό πλήθος των συναρτήσεων είναι σχεδόν ίδιες, κυμαίνονται ανάμεσα σε 23 με 22 ζευγάρια.

Το **SEE**(***sum of square error***) σε σχέση με την πραγματική ομοιότητα είναι το ακόλουθο:

Για 10 συναρτήσεις: 58018.59273139535
Για 50 συναρτήσεις: 18556.71379723621
Για 100 συναρτήσεις: 32542.80526484124
Για 200 συναρτήσεις: 24113.547530400258

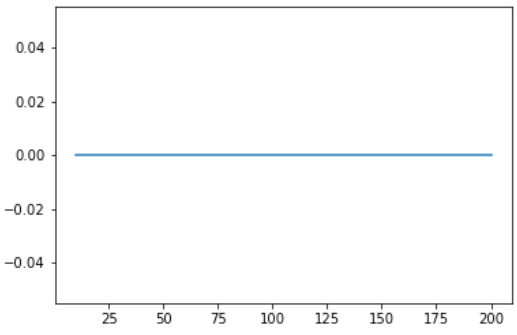


Συμπεράσματα από το γράφημα: Όταν οι συναρτήσεις είναι 50 τότε έχουμε την μικρότερη απόκλιση σε σύγκριση με την πραγματική ομοιότητα, ενώ όταν έχουμε 10 συναρτήσεις έχουμε την μεγαλύτερη απόκλιση. Όσο αυξάνει το πλήθος των συναρτήσεων το see κυμαίνεται σε χαμηλά επίπεδα.

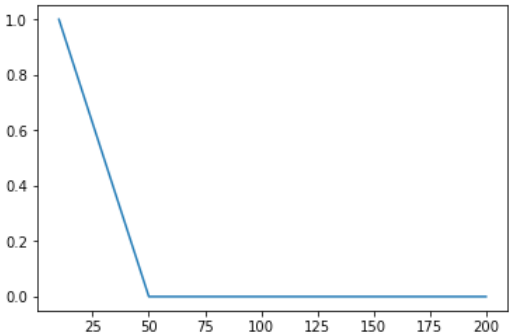
Υπολόγισα το πλήθος των ***false positives*** και τα ***false negatives*** και τα αποτελέσματα μαζί μα τα γραφήματα είναι τα ακόλουθα:

	10	**50**	**100**	**200**
False positives (FP)	0	0	0	0
False negatives (FN)	1	0	0	0

Γράφημα για τα fp



Γράφημα για τα fn

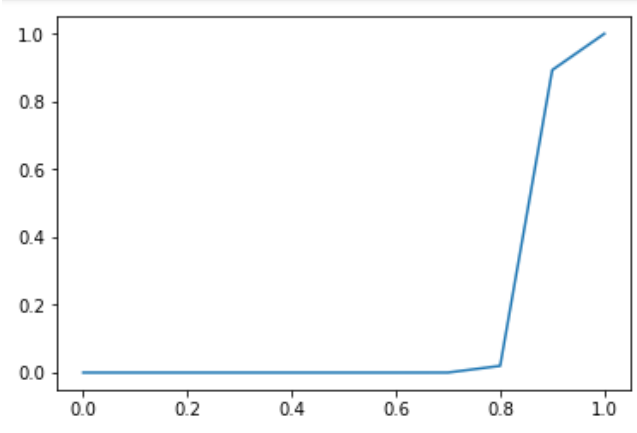


Συμπεράσματα για τα γραφήματα: Οι τιμές τους για τα διάφορα πλήθη των συναρτήσεων κυμαίνονται κοντά στο 0 και για τα fp και για τα fn.

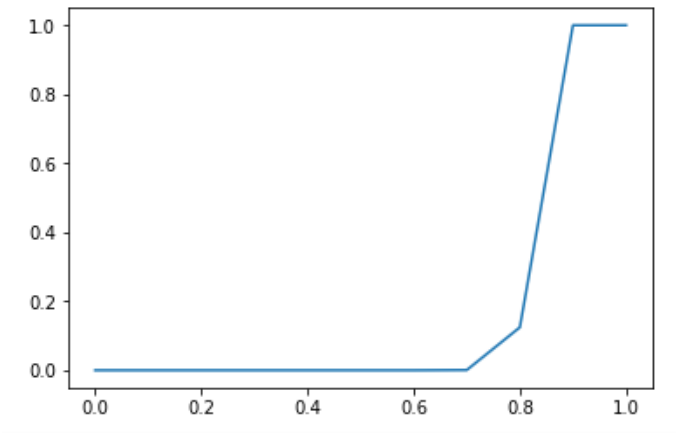
LSH

Θέλω να βρω ποιες είναι οι καλύτερες τιμές των ***b*** και ***r*** ώστε από την γραφική παράσταση της συνάρτησης "$1 - (1 - s^r)^b$" το σημείο όπου γίνεται η μεγάλη διαφορά να ξεκινάει από την τιμή 0.8. Κάποιες γραφικές παραστάσεις για διάφορες τιμές b και r είναι οι ακόλουθες:

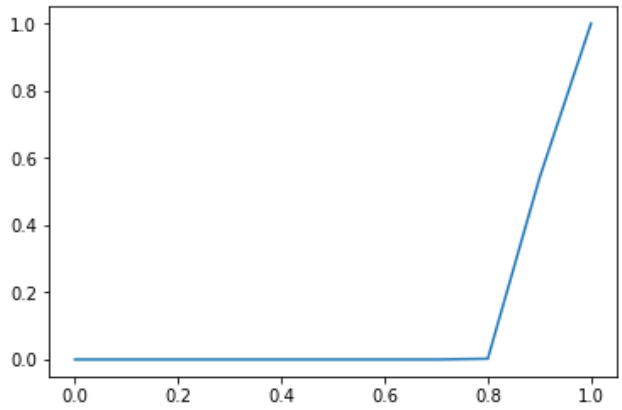
Για b = 100 και r = 40:



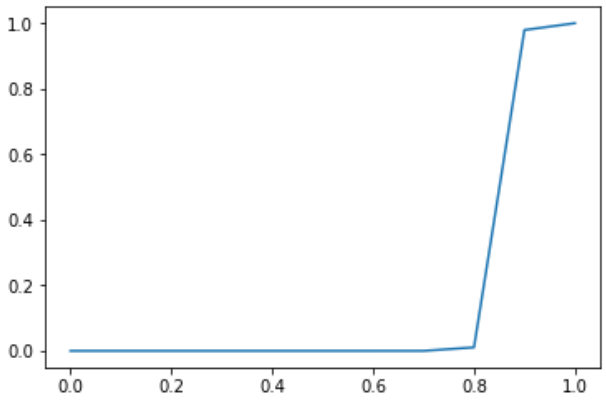
Για b = 1000 και r = 40:



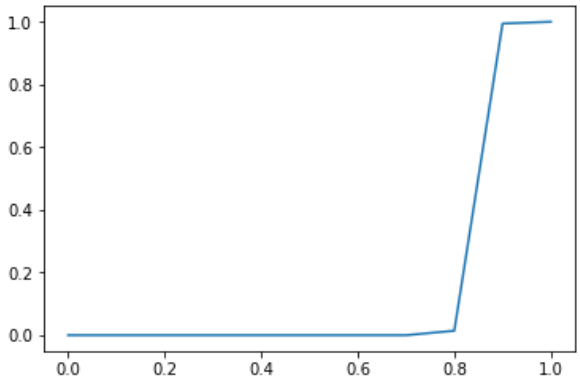
Για b = 100 και r = 50:



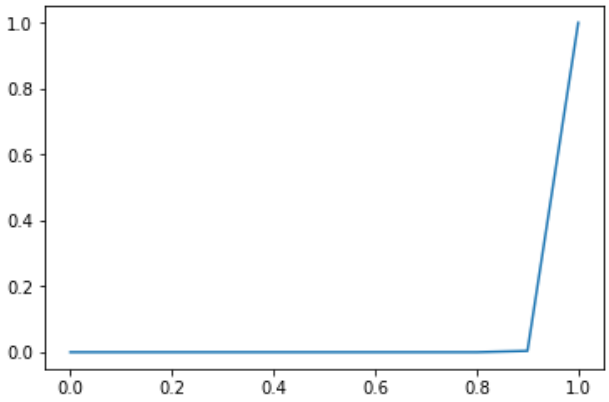
Για b = 700 και r = 50:



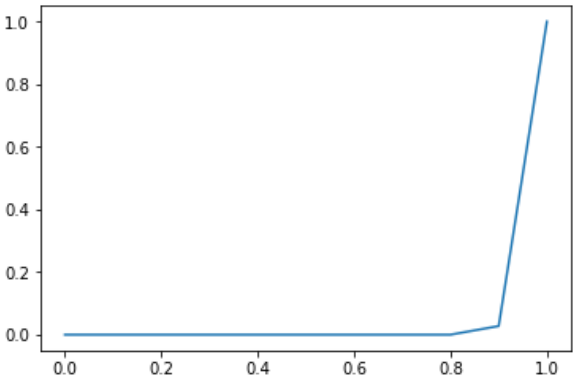
Για b = 1000 και r = 50:



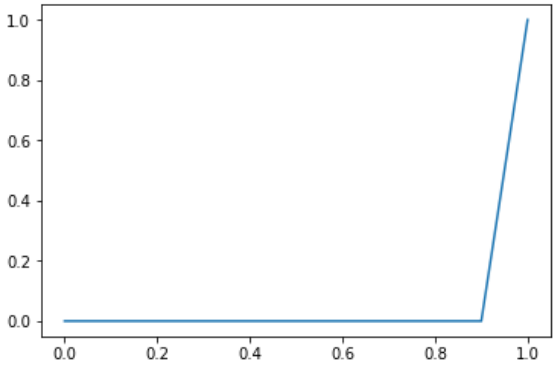
Για b = 100 και r = 100:



Για b = 1000 και r = 100:



Για b = 500 και r = 200:



Υλοποίηση του LSH:

Όπως και πιο πάνω στον κώδικα καλώ την συνάρτηση "***findHashSignatures()***" από την οποία έχω τα αποτελέσματα των συναρτήσεων. Έπειτα καλώ την συνάρτηση "***createTableForHashes(r)***" που αρχικοποιεί τον πίνακα με τις υπογραφές στο άπειρο. Γεμίζω τον πίνακα και φτιάχνω τον τελικό πίνακα με τις υπογραφές. Για κάθε χρήστη παίρνω τις υπογραφές του που έχουν προκύψει από τις r hash συναρτήσεις. Τις βάζω σε ένα λεξικό και κρατάω σαν κλειδί μία λίστα με τις τιμές των υπογραφών και σαν τιμή του κλειδιού τον χρήστη του. Οπότε είναι λογικό κάποιοι χρήστες να έχουν την ίδια λίστα με υπογραφές οπότε και θα είναι σαν τιμή στο ίδιο κλειδί. Αν ένα κλειδί έχει περισσότερους από έναν χρήστες τότε σημαίνει ότι αυτοί οι χρήστες(τα ζευγάρια τους) έχουν ομοιότητα > 0.8. Έπειτα βρίσκω το πλήθος όλων αυτών των ζευγαριών και τα false positives και false negatives. Τα αποτελέσματα που βρήκα για διάφορες τιμές των b και r είναι τα ακόλουθα:

Για b = 100 και r = 40:

Πλήθος ζευγαριών: 21
(Able_49,Doug_39), (Able_49,Earl_59), (Able_49,Bart_39), (Doug_39,Earl_59), (Doug_39,Bart_39), (Earl_59,Bart_39), (Doug_39,Chad_59), (Doug_39,ggeett37aaa), (Earl_59,Chad_59), (Earl_59,ggeett37aaa), (Chad_59,ggeett37aaa), (Able_49,Chad_59), (Chad_59,Bart_39), (Forprinciple16,goheels_70), (Able_49,ggeett37aaa), (Doug_39,dansch2002), (Earl_59,dansch2002), (ggeett37aaa,Bart_39), (Able_49,dansch2002), (Chad_59,dansch2002), (dansch2002,Bart_39)
false positives: 0
false negatives: 2

Για b = 500 και r = 50:

Πλήθος ζευγαριών: 20
(Able_49,Doug_39), (Able_49,Earl_59), (Doug_39,Earl_59), (Able_49,Bart_39), (Doug_39,Chad_59), (Earl_59,Chad_59), (Doug_39,Bart_39), (Forprinciple16,goheels_70), (Doug_39,ggeett37aaa), (Earl_59,ggeett37aaa), (Chad_59,ggeett37aaa), (Earl_59,Bart_39), (Doug_39,dansch2002), (Able_49,Chad_59), (Chad_59,Bart_39), (Chad_59,dansch2002), (Able_49,ggeett37aaa), (ggeett37aaa,Bart_39), (Earl_59,dansch2002), (Able_49,dansch2002)

false positives: 0
false negatives: 3

Για b = 100 και r = 100:

Πλήθος ζευγαριών: 10
(Able_49,Doug_39), (Able_49,Earl_59), (Able_49,Chad_59), (Doug_39,Earl_59), (Doug_39,Chad_59), (Earl_59,Chad_59), (Forprinciple16,goheels_70), (Able_49,Bart_39), (Doug_39,Bart_39), (Doug_39,ggeett37aaa)
false positives: 0
false negatives: 13

Με βάση τις γραφικές παραστάσεις πιο πάνω και τα αποτελέσματα θα κοιτάξω τα tweets των χρηστών για την περιπτώση b = 100 r = 40 γιατί αυτό βρήκε τα περισσότερο κοινά ζευγάρια σε σχέση με τον αριθμό 23 όπου είναι συνολικά. Τώρα θα πρέπει να αποφασίσω ποιοι χρήστες είναι bots και ποιοι είναι trolls.

Bots → Able_49, ggeett37aaa, Bart_39

Trolls → Doug_39, Earl_49, goheels_70, dansch2002

Ερώτηση 3(Συστήματα συστάσεων)

Θέλουμε να έχουμε τα κλαδεμένα δεδομένα της Ερώτησης 2, την διαδικασία την έχω αναφέρει στην Ερώτηση 2 για το πως παίρνω τα κλαδεμένα δεδομένα. Συνεχίζω στο μέρος όπου πρέπει να διαλέξω 10% από τους χρήστες. Ανάμεσα στους 5360 χρήστες που μου έχουν απομείνει το 10% αυτών είναι 536. Διαλέγω ανάμεσα στους αριθμούς [0, 5363] ***536 αριθμούς*** τυχαία οι οποίοι αντιπροσωπεύουν έναν χρήστη. Τους αριθμούς αυτούς τους κρατάω μέσα στην λίστα "***chooseUsers***". Σε περίπτωση που επιλεγεί κάποιος ίδιος αριθμός αυτό το διαχειρίζομαι μέσα σε μία while και ελέγχω κάθε φορά πόσους διαφορετικούς αριθμούς/χρήστες έχω διαλέξει ώστε στο τέλος να είναι σίγουρα το 10% αυτών. Μέσα στην λίστα "***nameUsersList***" έχω όλα τα ονόματα των χρηστών. Έπειτα για κάθε έναν από τους χρήστες που έχουν επιλεγεί τυχαία, παίρνω το σύνολο από τα hashtags/handles που έχουν χρησιμοποιήσει. Ο τρόπος με τον οποίο διαλέγω αυτό το hashtag ή handle είναι να πάρω το μέγεθος από τα δεδομένα που έχει χρησιμοποιήσει και να διαλέξω έναν αριθμό ανάμεσα σε αυτό το πλήθος. Φτιάχνω ένα καινούριο λεξικό για όλους τους χρήστες με κλειδί το όνομα του χρήστη και τιμή μία λίστα με τα hashtags/handles που χρησιμοποιεί χωρίς αυτά που αφαιρέσαμε πριν. Μερικοί από τους χρήστες για τους οποίους θα ελέγχουμε αν τελικά οι αλγόριθμοι βρουν τα hashtag/handle που αφαιρέθηκε είναι οι ακόλουθοι:

For users "127bama": @hotlinejosh
For users "Stilgar3": @fbi
For users "Retiree07": @erictrump
For users "theUKtoday": #potus
For users "JPBatl": @davidaxelrod
For users "ScarpMichael": @foxnews
For users "BirdingTrip": @politico
For users "midsgt": @cnnpolitics
For users "stinkpickle1976": @ericbolling
For users "lindaantonO": @detroitnews
For users "headkel": @drudge_report
For users "jnjsmom": @martinomalley
For users "karindarby": #maga
For users "arjanomics": @katyturnbc
For users "EnoughisEnoug13": @seanhannity
For users "brobans": @danney_williams
For users "get": @chelseaclinton
For users "LFS7": @robbymook
For users "VenableAaron": @truethevote
For users "jbreesenaz": @joebiden

For users "pfmcmanus36": #kellyfile For users "jakub_lisa": #corruptmedia

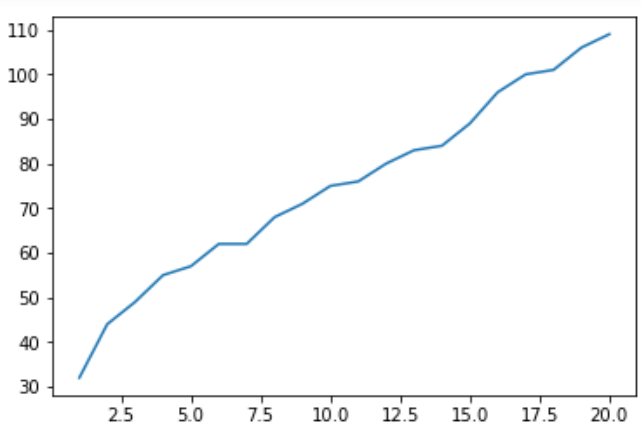
Αλγόριθμοι

1. MostPopular

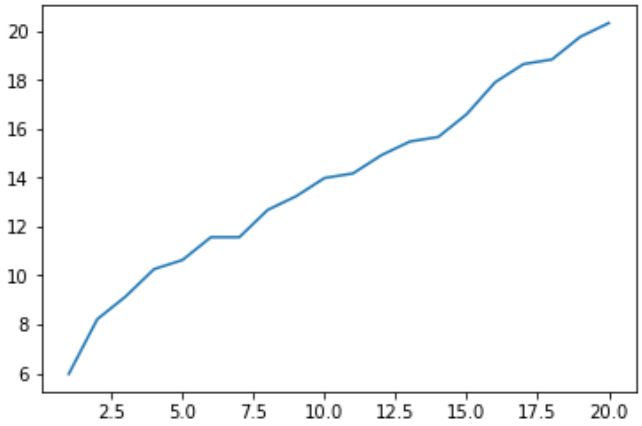
Έχω φτιάξει την συνάρτηση "***def mostPopular(name, k)***" η οποία δέχεται σαν όρισμα το όνομα του χρήστη από τον οποίο έχουμε αφαιρέσει κάποιο hashtag/handle και k είναι πόσα από τα καλύτερα hashtags/handles θα επιστρέψουμε στον χρήστη. Η συνάρτηση αυτή φτιάχνει ένα λεξικό "***dictionaryToFindMax***" όπου σαν κλειδί έχει τα hashtags/handles τα οποία δεν χρησιμοποιεί ο χρήστη με όνομα name και σαν τιμή το πλήθος των χρηστών συνολικά που τα χρησιμοποιούν. Αυτό το πλήθος το έχω κρατήσει σε ένα λεξικό με όνομα "***finalHashtagsHandles***" και τιμή το πλήθος των χρηστών που τα χρησιμοποιεί. Έπειτα κάνω ταξινόμηση το λεξικό "***dictionaryToFindMax***" σε φθίνουσα σειρά και παίρνω τα k καλύτερα και τα επιστρέφω στον χρήστη.
Αφού επιστραφεί το αποτέλεσμα στον χρήστη και για όλους τους χρήστες υπολογίζω τις επιτυχίες για τις οποίες βρήκα τα εξής για τιμές k από [1, 20]:

Plithos epityxiwn: [32, 44, 49, 55, 57, 62, 62, 68, 71, 75, 76, 80, 83, 84, 89, 96, 100, 101, 106, 109]

Πλήθος



Ποσοστό %



Συμπέρασμα από το γράφημα: Για μικρές τιμές του k το ποσοστό επιτυχίας για τον αλγόριθμο mostPopular η επιτυχία είναι χαμηλή κοντά στο 6% ενώ για μεγαλύτερες τιμές του k η επιτυχία πλησιάζει κοντά στο 21%. Και για άλλες περιπτώσεις που το έτρεξα(κάπου στις 10) πάλι το ποσοστό για k = 20 ήτανε κοντά στο 20% - 25%.

2. User-based Collaborative Filtering(UCF)

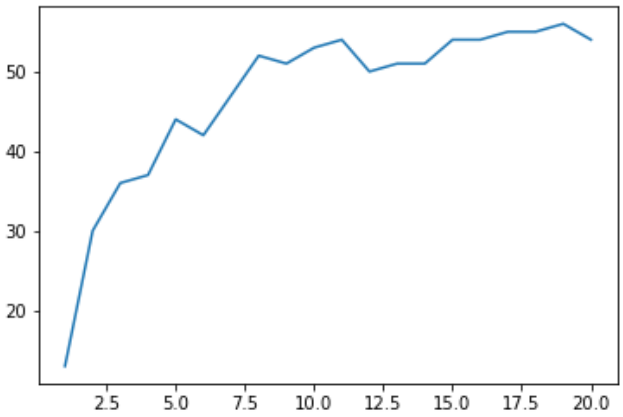
Έχω φτιάξει την συνάρτηση "***def jaccardSimilarity(x, y)***" η οποία δέχεται σαν όρισμα δύο σύνολα και υπολογίζει την jaccard ομοιότητα τους και την επιστρέφει. Υπάρχει η συνάρτηση "***def ucf(name, m)***" η οποία υλοποιεί τον αλγόριθμο. Παίρνει σαν όρισμα το όνομα του χρήστη για τον οποίο θέλουμε να βρούμε τους m πιο όμοιους με αυτόν χρήστες. Φτιάχνω ένα λεξικό "***jsDictionary***" το οποία κρατάει μέσα το όνομα του χρήστη με τον οποία έκανε σύγκριση και σαν τιμή την js ομοιότητα τους. Κρατάω τους m καλύτερους μέσα στο "***mSimilarUsers***" αλλά επειδή υπολογίζω και την ομοιότητα του χρήστη για τον οποίο ψάχνουμε με τον εαυτό του πάντα θα βρίσκεται στην πρώτη θέση οπότε παίρνω τους όμοιους με αυτόν χρήστες από την θέση 1 και όχι από την 0. Στην λίστα "***geitones***" κρατάω τα

ονόματα από τους όμοιους χρήστες και στην λίστα “***similarity***” την js ομοιότητα τους. Για όλα τα hashtags/handles φτιάχνω ένα λεξικό με όνομα “***scoreForEveryHashtagHandle***” και για κάθε κλειδί αρχικοποιώ την τιμή του στο 0. Για τον κάθε χρήστη από τους όμοιους παίρνω στην μεταβλητή με όνομα “***hashtagHandleApoGeitones***” τα hashtags/handles που έχουν χρησιμοποιήσει. Για κάθε ένα από αυτά πηγαίνω στο αντίστοιχο κλειδί στο λεξικό “***scoreForEveryHashtagHandle***” και του προθέτω την js ομοιότητα του χρήστη αυτού. Τέλος κάνω φθίνουσα ταξινόμηση στο λεξικό αυτό και παίρνω τα k καλύτερα hashtags/handles και τα επιστρέφω στον χρήστη.

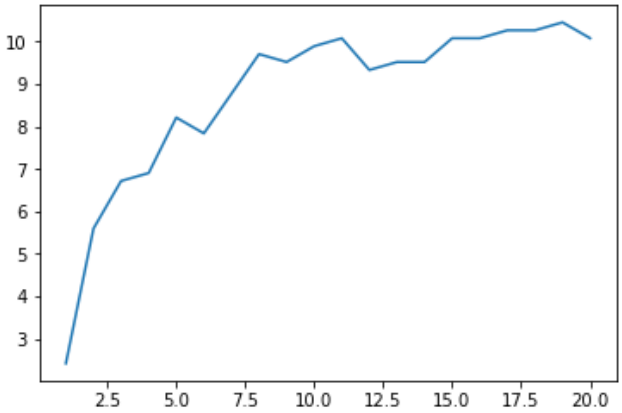
Καλώ την συνάρτηση “***ucf()***” για κάθε έναν χρήστη από αυτούς που έχω αφαιρέσει κάποιο hashtag/handle και κάθε έναν από αυτούς βρίσκω m διαφορετικούς όμοιους από [1, 20]. Σχετικά με το ποσοστό επιτυχίας τα αποτελέσματα φαίνονται στις παρακάτω γραφικές παραστάσεις:

Plitos epityxiwn: [13, 30, 36, 37, 44, 42, 47, 52, 51, 53, 54, 50, 51, 51, 54, 54, 55, 55, 56, 54]

Πλήθος



Ποσοστό%



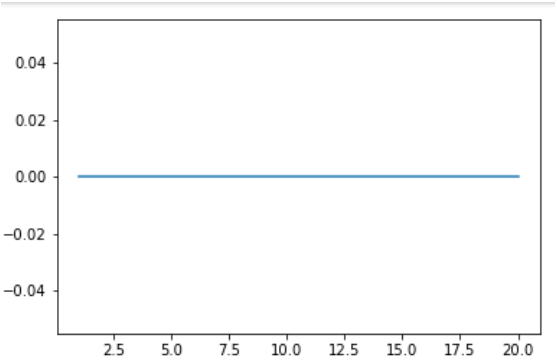
Συμπέρασμα από το γράφημα: Με τον αλγόριθμο ucf το ποσοστό επιτυχίας για διάφορες τιμές του m κυμαίνονται από [2% - 10%]. Και για άλλες περιπτώσεις που το έτρεξα(κάπου στις 10) πάλι το ποσοστό επιτυχίας ήτανε σε αυτές τις τιμές.

3. Item-based Collaborative Filtering (ICF)

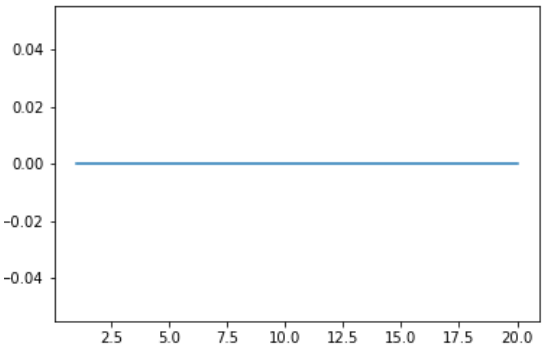
Αρχικά για κάθε χρήστη που του έχω αφαιρέσει κάτι θα έπρεπε να βρω όλα τα js των hashtags/handles του που χρησιμοποιεί με όλα όσα δεν χρησιμοποιεί. Με μία δοκιμή αυτό έπαιρνε αρκετό χρόνο οπότε οπότε βρίσκω τα js όλων των hashtags/handles όλων αυτών που δεν χρησιμοποιούν όλοι οι χρήστες με όλα εκείνα που χρησιμοποιούν. Ίσως να υπάρχει κάποια απόκλιση στο αποτέλεσμα αλλά θα είναι πολύ μικρή.
Έχω μία συνάρτηση με όνομα “***def icf(lis)***” η οποία υπολογίζει τις ομοιότητες όλων με όλα(αυτά που θέλουμε να υπολογίσουμε βέβαια). Το lis Που δέχεται σαν όρισμα η συνάρτηση είναι τα hashtags/handles που έχουν χρησιμοποιήσει όλοι οι χρήστες από τους για τους οποίους έχουμε αφαιρέσει κάποιο hashtag/handle. Οπότε αυτή η συνάρτηση μου επιστρέφει ένα λεξικό με κλειδί ένα hashtag/handle που δεν έχουν χρησιμοποιήσει οι χρήστες και σαν τιμές ένα άλλο λεξικό με κλειδί τα hashtags/handles που έχουν χρησιμοποιήσει και την ομοιότητα τους με το κλειδί. Αυτό το λεξικό το επιστρέφω στην μεταβλητή “***dicti***”.
Έπειτα για κάθε έναν από τους χρήστες που ψάχνουμε τα k καλύτερα για αυτόν καλώ την συνάρτηση “***def icf2(allDictionary, name, m)***” η οποία ταξινομεί το εσωτερικό λεξικό “***allDictionary***” που δέχεται σαν όρισμα και και παίρνω τα m πιο όμοια για κάθε ένα από τα κλειδιά του λεξικού “***allDictionary***”.

Έπειτα υπολογίζω το score για κάθε ένα από αυτά τα hashtags/handles με τον τύπο που μας δίνεται και κρατάω τα k καλύτερα(k = 10).
Το ποσοστό επιτυχίας του αλγορίθμου αυτού φαίνεται στις παρακάτω γραφικές παραστάσεις:
Plitos epityxiwn: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Πλήθος



Ποσοστό%



Συμπεράσματα από το γράφημα: Δεν ήτανε καθόλου επιτυχημένος αυτός ο αλγόριθμος. Να αναφέρω ότι για ένα άλλο παράδειγμα όπου το είχα τρέξει είχε επιτυχία 9% μόνο για m = 20.

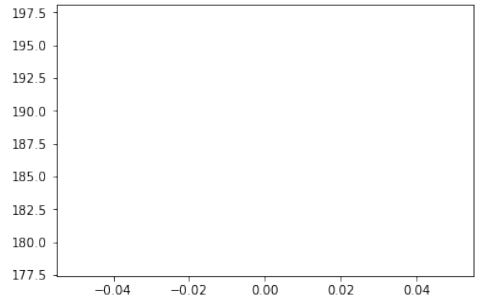
4. Singular Value Decomposition (SVD)

Αρχικά φτιάχνω τον πίνακα "***myArray***" με 0/1 για όλους τους χρήστες και τα hashtags/handles. Στον άξονα Y κρατάω τα hashtags/handles και στον άξονα X τους χρήστες.

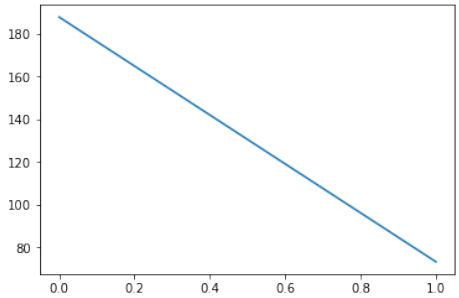
```
[[0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 1 0 ..., 0 0 0]
 ...,
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]
 [0 0 0 ..., 0 0 0]]
(2684, 5360)
```

Βρίσκω τα U, S, V από τον svd με την ***βιβλιοθήκη linalg.svd***. Κάνω τις γραφικές παραστάσεις για διάφορα m-rank mE[1, 20] για να δω ποιο m είναι "καλύτερο". Οι γραφικές παραστάσεις που παίρνω είναι οι ακόλουθες:

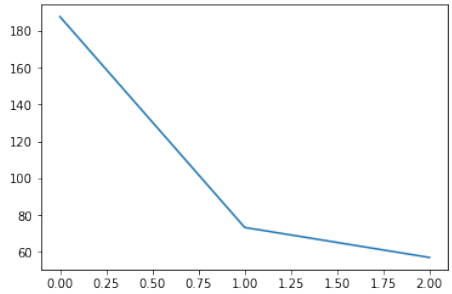
m: 1



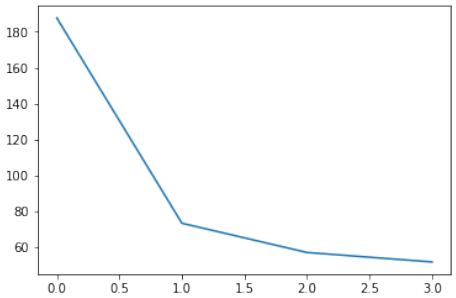
m: 2



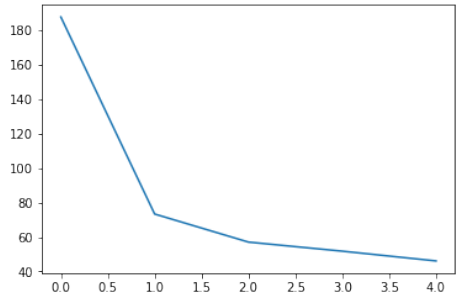
m: 3



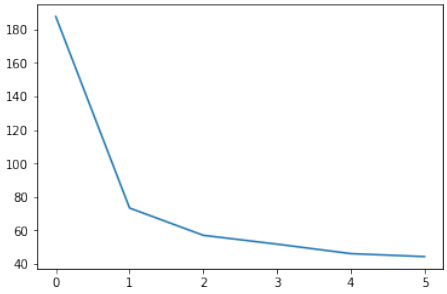
m: 4



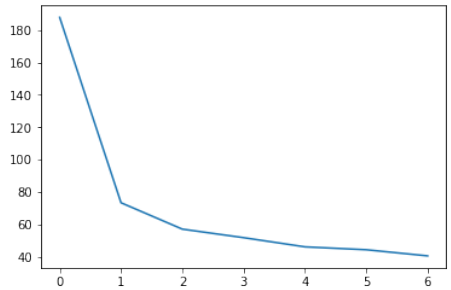
m: 5



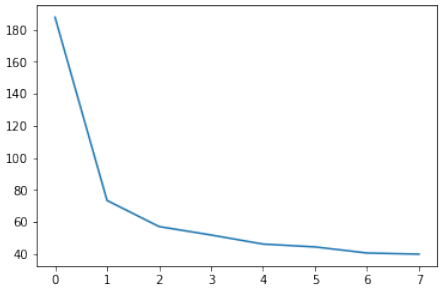
m: 6



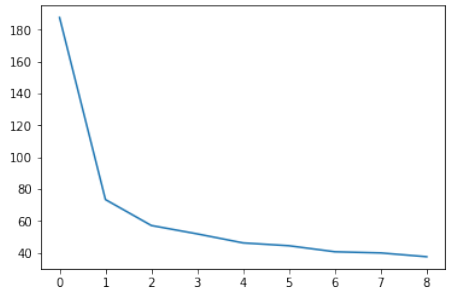
m: 7



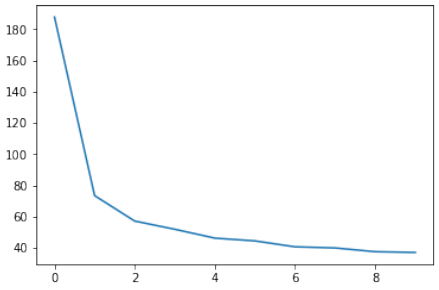
m: 8



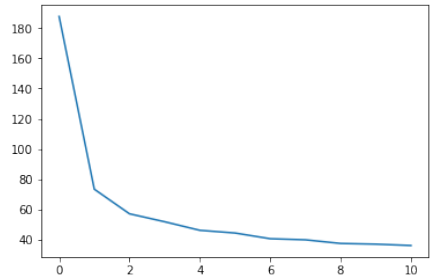
m: 9



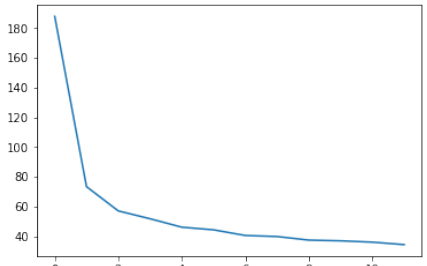
m: 10



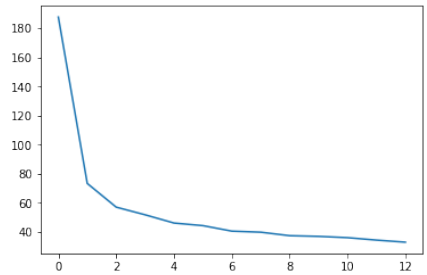
m: 11



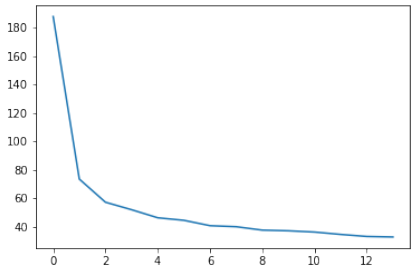
m: 12



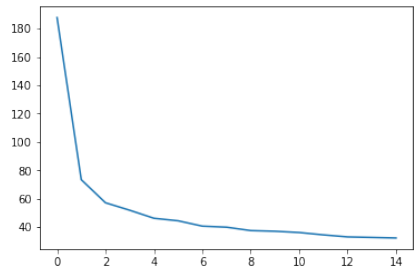
m: 13



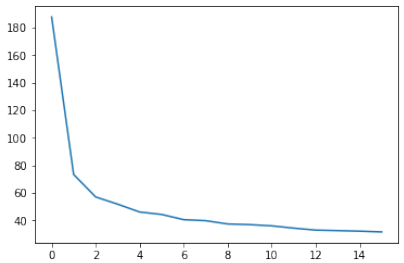
m: 14



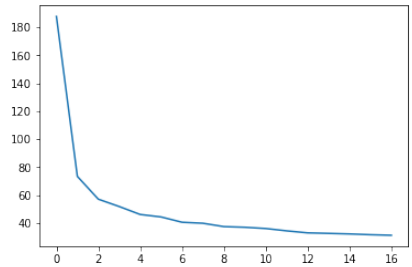
m: 15



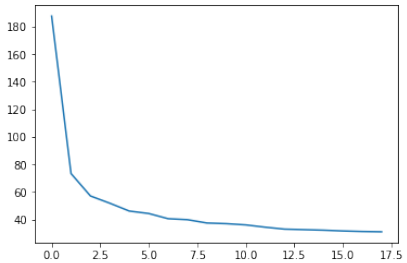
m: 16



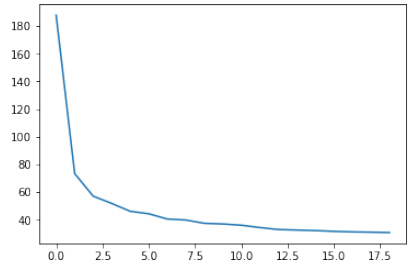
m: 17



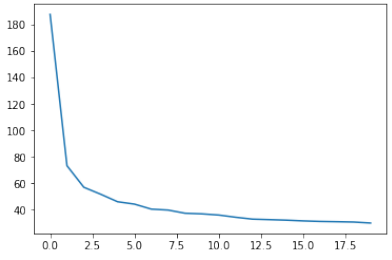
m: 18



m: 19



m: 20

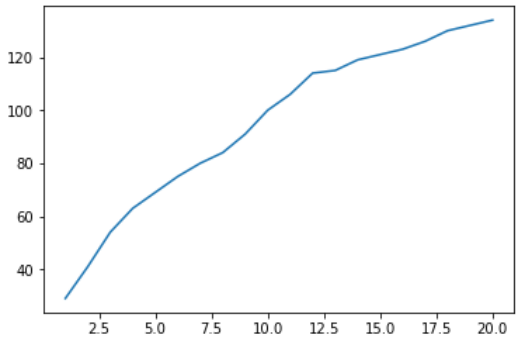


Συμπεράσματα από τις γραφικές: Για m = 3 ή m = 4 φαίνεται καλύτερα σε ποιο σημείο υπάρχει αυτό το "γόνατο". Οπότε θα διαλέξω m = 3.

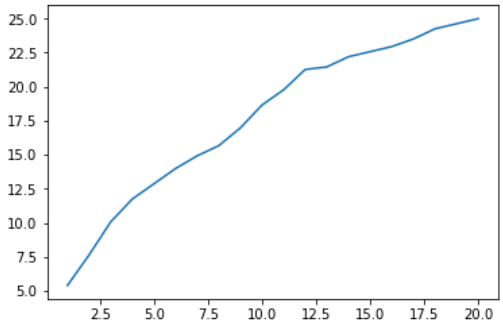
Έχω φτιάξει την συνάρτηση "***def svd(name, number, k, Mr)***" η οποία δέχεται σαν όρισμα το όνομα του χρήστη που θέλουμε να του επιστρέψουμε τα k καλύτερα αποτελέσματα, τον αριθμό του για να είναι πιο εύκολο να πάρω κατευθείαν την στήλη με όλα τα hashtags/handles από τον πίνακα Mr και να διαλέξω τα k καλύτερα και να τα επιστρέψω στον χρήστη. Ο πίνακας Mr υπολογίζεται μία φορά και είναι ο $M = USV^T \rightarrow Mr = UrSrVr^T$. Τον αλγόριθμο ***svd()*** τον καλώ για κάθε χρήστη και το ποσοστό επιτυχίας του είναι το ακόλουθο:

Plitos epityxiwn: [29, 41, 54, 63, 69, 75, 80, 84, 91, 100, 106, 114, 115, 119, 121, 123, 126, 130, 132, 134]

Πλήθος



Ποσοστό%



Συμπεράσματα από το γράφημα: Βρίσκει το 1/4 περίπου των hashtags/handles που αφαιρέσαμε.

Από όλους τους αλγορίθμους καλύτερα αποτελέσματα μου δίνει:

1.SVD → 2.mostPopular → 3.UCF → 4.ICF