



ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Εργασία 2

ΜΙΧΟΥ ΓΕΩΡΓΙΑ 3828

Η εργασία εκπονήθηκε από την άνωθεν φοιτήτρια του τμήματος Πληροφορικής του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, στα πλαίσια του μαθήματος «Νευρωνικά Δίκτυα - Βαθιά Μάθηση» κατά τη διάρκεια του ακαδημαϊκού έτους 2023-2024.

1. Γενικά

Η παρακάτω εργασία αφορά την δημιουργία μιας μηχανής διανυσμάτων υποστήριξης (Support Vector Machine) που χρησιμοποιείται για την κατηγοριοποίηση δύο κλάσεων. Χρησιμοποιείται η βάση δεδομένων cifar-10 όπου επιλέγονται από εκείνη δύο διαφορετικά ζεύγη κλάσεων. Η εργασία έχει υλοποιηθεί στη γλώσσα προγραμματισμού Python, στο περιβάλλον google colab και επισυνάπτονται 2 αρχεία .ipynb, ένα που αφορά υλοποίηση από την αρχή (my_svm) και ένα που χρησιμοποιείται έτοιμο το svm από την βιβλιοθήκη sklearn (sklearnSVM).

2. Κώδικας - Υλοποίηση

Α) Βάση δεδομένων και επεξεργασία αυτών: Στην αρχή, φορτώνεται η βάση δεδομένων από το keras και επιλέγονται 2 κλάσεις της οι οποίες θα χρησιμοποιηθούν μετέπειτα για δυαδική ταξινόμηση μέσω του svm. Οπότε, δημιουργούνται δυαδικές ετικέτες όπου τα στοιχεία που αντιστοιχούν στην επιλεγμένη κλάση (class_labels[0]) χαρακτηρίζονται ως 1 και τα υπόλοιπα ως -1. Έπειτα, τα δεδομένα μετασχηματίζονται σε 1-d arrays και κανονικοποιούνται. Λόγω χρονικής πολυπλοκότητας, επιλέγονται συγκεκριμένα υποσύνολα από τα δεδομένα εκπαίδευσης και ελέγχου διατηρώντας την αναλογία μεταξύ αυτών όπως υπάρχει στη βάση. Ακολουθεί η κλάση SVM η οποία υλοποιεί τον αλγόριθμο με τον οποίο θα γίνει η ταξινόμηση στα δεδομένα των κλάσεων. Ωστόσο, λόγω χρονικής πολυπλοκότητας και δυσκολίας, παρακάτω χρησιμοποιείται στα δεδομένα και έτοιμο το svm από την βιβλιοθήκη sklearn.

Β) Υλοποίηση του αλγορίθμου: Αρχικά, έχουν δημιουργηθεί κάποιες βιοηθητικές μέθοδοι και παράμετροι που συμβάλλουν στην εκπαίδευση της μηχανής. Αρχίζοντας από τις παραμέτρους, έχουμε:

- 1) kernels (dictionary): περιλαμβάνει 2 μεθόδους για γραμμικό πυρήνα και πολυωνυμικό αντίστοιχα. Αυτές ορίζονται ως kernel_linear και kernel_polyomial όπου στην τελευταία ορίζεται και ο βαθμός που θα έχει το πολυώνυμο

- 2) degree: ο βαθμός του πολυώνυμου (τίθεται πάντα None στο linear kernel)
- 3) max_iter: ορίζεται ο μέγιστος αριθμός επαναλήψεων κατά την εκπαίδευση
- 4) kernel_type: ο τύπος πυρήνα που θα χρησιμοποιήσουμε
- 5) C: η παράμετρος κανονικοποίησης
- 6) epsilon: Κατώφλι σύγκλισης. Εάν η μεταβολή των τιμών άλφα είναι κάτω από αυτό το όριο, ο βρόχος εκπαίδευσης σταματά.
- 7) alpha: οι πολλαπλασιαστές Lagrange που έχουν τον ρόλο των συντελεστών που αντιστοιχίζονται σε κάθε παράδειγμα εκπαίδευσης
- 8) support_vectors: πίνακας στον οποίον αποθηκεύουμε τα διανύσματα υποστήριξης
- 9) w, b: τα διάνυσμα βάρους και bias αντίστοιχα

Οι μέθοδοι που υλοποιούνται πριν την εκπαίδευση:

- 1) set_params: εδώ ορίζουμε τις περισσότερους παραμέτρους από όσες προαναφέρθηκαν. Δεν έχουν καθοριστεί συγκεκριμένες τιμές σε αυτές ώστε να δοκιμαστεί η λειτουργικότητα του svmt με διαφορετικές τιμές.
- 2) linear_kernel: εφαρμόζεται για τον υπολογισμό του γινομένου δύο διανυσμάτων εισόδου, χαρακτηριστικό του γραμμικού πυρήνα. Η συνάρτηση απόφασης χρησιμοποιεί αυτόν τον γραμμικό πυρήνα για τον προσδιορισμό των τιμών απόφασης.
- 3) kernel_polyomial: υπολογίζει το γινόμενο δύο διανυσμάτων εισόδου λαμβάνοντας υπόψιν τον βαθμό του πολυώνυμου, χαρακτηριστικό των πολυωνυμικών πηρήνων.
- 4) calc_b: υπολογίζει την τιμή του bias παίρνοντας την διαφορά των ετικετών γ με το γινόμενο του w και των

δεδομένων εισόδου X. Η μέση τιμή αυτής της διαφοράς ανατίθεται στο bias.

- 5) calc_w: μέθοδος υπολογισμού τιμών των βαρών με βάση τους πολλαπλασιαστές Lagrange. Πρώτα υπολογίζεται το γινόμενο αυτών με τις ετικέτες γ και μετά το dot προϊόν αυτού με τα δεδομένα εισόδου X. Κάθε στοιχείο του w αντιστοιχεί σε ένα χαρακτηριστικό στο χώρο εισόδου και οι τιμές του w καθορίζονται από τους πολλαπλασιαστές Lagrange και τις ετικέτες κλάσης.
- 6) h: υπολογίζει το πρόσημο της συνάρτησης απόφασης, υποδεικνύοντας τις προβλεπόμενες ετικέτες κλάσης.
- 7) E: υπολογίζει τον όρο σφάλματος για ένα δεδομένο εισόδου, παίρνοντας την διαφορά μεταξύ της προβλεπόμενης και πραγματικής τιμής.
- 8) decision_boundary: υπολογίζει το όριο απόφασης με βάση τα w, b. Επιστρέφει το όριο με την προϋπόθεση πως το w είναι μονοδιάστατος πίνακας.
- 9) compute_L_H: υπολογίζει τα L, H που μέσα στην fit χρησιμοποιείται για την επιλογή νέων πολλαπλασιαστών Lagrange. Αναλυτικά, τα i, j που υπάρχουν στα alrha ορίζουν σε ποιο παράδειγμα αναφέρεται το καθένα, όπως και στα γ. Αν τα yi, yj έχουν διαφορετικές ετικέτες κλάσης, τότε θα έχουμε $L = \max(0, a_j' - a_i')$ και $H = \min(C, C - a_i' + a_j')$. Αν όμως έχουν τις ίδιες, $L = \max(0, a_i' + a_j' - C)$ και $H = \min(C, a_i' + a_j')$.
- 10) get_rnd_int: δημιουργεί έναν τυχαίο ακέραιο (χρησ. στην fit)
- 11) decision_function: υπολογίζει την συνάρτηση απόφασης για το σύνολο εισόδου X. Ορίζεται ως το τετραγωνικό γινόμενο του διανύσματος w με τον πίνακα δεδομένων εισόδου X, συν τον όρο bias.

Ακολουθεί η μέθοδος fit όπου αποτελεί την βάση στην υλοποίηση του svm. Όπως και οι παραπάνω μέθοδοι, η fit έχει σχεδιαστεί με σκοπό να εκπαιδεύει με την λογική που χρησιμοποείται στον αλγόριθμο SMO. Ξεκινά αρχικοποιώντας τις τιμές σε πολλές από τις παραμέτρους που προαναφέρθηκαν καθώς και σε κάποιες που δημιουργούνται για να υπολογίσουμε τις σωστές και λανθασμένες ταξινομήσεις. Μέσα σε έναν βρόγχο που δεν ξεπερνά το μέγιστο πλήθος επαναλήψεων αρχίζει η εκπαίδευση. Σε κάθε επανάληψη διατηρείται ένα αντίγραφο από τους πολλαπλασιαστές Lagrange που χρησιμοποιούνται (alpha_prev). Στην πρώτη επανάληψη προσδιορίζονται και αρχικοποιούνται τα διανύσματα υποστήριξης με βάση μη μηδενικούς και μη καλυμμένους πολλαπλασιαστές. Για κάθε επανάληψη, εκτελούνται τα παρακάτω βήματα: η μέθοδος επιλέγει τυχαία ένα ζεύγος παραδειγμάτων i , j και υπολογίζει την αντίστοιχη τιμή πυρήνα (k_{ij}). Ανακτά τους υπάρχοντες πολλαπλασιαστές Lagrange για το επιλεγμένο ζεύγος και υπολογίζει τα όρια (L , H) χρησιμοποιώντας τη μέθοδο compute_L_H. Το διάνυσμα βάρους (w) και ο όρος μεροληψίας (b) λαμβάνονται χρησιμοποιώντας τις μεθόδους calc_w και calc_b. Για το επιλεγμένο ζεύγος, γίνεται εκτίμηση των σφαλμάτων Ei και Ej και έπειτα ενημερώνονται κατάλληλα οι πολλαπλασιαστές Lagrange. Με βάση αυτά, ενημερώνονται έπειτα τα διανύσματα υποστήριξης (support vectors). Στη συνέχεια η μέθοδος αποτιμά τις προβλέψεις που κάνει το μοντέλο για τα συγκεκριμένα παραδείγματα και ενημερώνει τους μετρητές για σωστές και λανθασμένες ταξινομήσεις. Ανά 5 επαναλήψεις, έχει οριστεί να εκτυπώνονται κάποια συγκεκριμένα στοιχεία με σκοπό να αξιολογηθεί η συμπεριφορά του μοντέλου. Αυτά είναι η διαφορά, πόσες σωστές ταξινομήσεις προέκυψαν και το ποσοστό αυτών σε σχέση με το συνολικό πλήθος παραδειγμάτων, τα αντίστοιχα για τις λανθασμένες ταξινομήσεις ενώ εκτυπώνονται και ορισμένα δείγματα αυτών. Αφού ολοκληρωθούν οι επαναλήψεις, εκτυπώνεται η απόδοση που πέτυχε το μοντέλο. Στο τέλος, η μέθοδος επιστρέφει τα διανύσματα υποστήριξης και τους πολλαπλασιαστές Lagrange.

Σε γενικές γραμμές, η fit έχει σχεδιαστεί ώστε να εκτελεί την εκπαίδευση του svm. Ενημερώνει επαναληπτικά τους πολλαπλασιαστές Lagrange, τα διανύσματα υποστήριξης και τις παραμέτρους του μοντέλου για να βρει το υπερεπίπεδο που διαχωρίζει καλύτερα τις κλάσεις στα δεδομένα εισόδου. Η διαδικασία περιλαμβάνει τυχαία επιλογή ζευγαριών, υπολογισμό σφαλμάτων και ενημέρωση των πολλαπλασιαστών Lagrange μέχρι τη σύγκλιση ή την επίτευξη του μέγιστου αριθμού επαναλήψεων. Η μέθοδος παρέχει επίσης πληροφορίες σχετικά με την πρόοδο της εκπαίδευσης μέσω τυπωμένων πληροφοριών.

Γ) Μοντέλο SVM και αξιολόγησή του: Δημιουργείται το model, αντικείμενο της κλάσης SVM και εκπαιδεύεται κάθε φορά με διαφορετικές παραμέτρους προκειμένου να εκτιμηθεί η συμπεριφορά του. Πριν την εκπαίδευση, αρχικοποιείται ο χρόνος ώστε να αποτιμηθεί ο συνολικός χρόνος αυτής. Έπειτα, εκτυπώνονται τα όρια απόφασης, τα διανύσματα υποστήριξης καθώς και οι προβλεπόμενες ετικέτες. Για παραπάνω εμβάθυνση, μετριέται η απόδοση στα δεδομένα εκπαίδευσης μέσω της accuracy_score καθώς και στα δεδομένα ελέγχου. Έπειτα, υπολογίζεται ο confusion matrix, δηλαδή ο πίνακας με τα πραγματικά θετικά, πραγματικά αρνητικά, λανθασμένα θετικά και λανθασμένα αρνητικά, τόσο για τα δεδομένα εκπαίδευσης αλλά και ελέγχου. Οι δύο τελευταίοι πίνακες προβάλλονται και σε γραφήματα για πληρότητα.

Δ) Πίνακας αποτελεσμάτων: Ο παρακάτω πίνακας περιέχει τις δοκιμές που έγιναν στο μοντέλο με στόχο την αξιολόγηση του. Έχουν χρησιμοποιηθεί διαφορετικές τιμές σε παραμέτρους του, συγκρίνοντας κάθε φορά ένα διαφορετικό χαρακτηριστικό του. Για πληρότητα, ακολουθούν εικόνες με τα αποτελέσματα που αποτυπώνονται στον πίνακα.

Size train data	Size test data	Kernel	C	Epsilon	max_iter	accuracy (1.train, 2.test)	Training time	Support vectors	Compare
500	100	Linear	1.0	0,001	100	48.4% 41.0%	6418.30	244	Value of epsilon
				0,01		36.8% 32.0%	6730.34	244	
200	50	Linear	1.0	0,001	100	51.50% 40.0%	542.43	104	Value of max iterations
				0.5	200	35.5% 28.0%	1125.97	103	
					50	51.50% 40.0%	276.45	104	
				2.0	100	36.5% 28.0%	568.16	97	Value of C
					100	51.50% 40.0%	563.24	106	
500	100	Polynomial - d = 1	1.0	0,001	100	38.6% 32.0%	6839.14	246	Degree
500	100	Polynomial - d = 2	1.0	0,001	100/100 (ends in 1)	48.8% 42.0%	64.91	125	Value of max iteration s
500	100	Polynomial - d = 3	1.0	0,001	100	48.8% 42.0%	64.91	125	
500	100	Polynomial - d = 1	1.0	0,01	100	34.6% 31.0%	13294.16	231	
500	100	Polynomial - d = 1	1.0	0,01	200	45.6% 37.0%	6636.54	244	

1η εκτέλεση :

```

Iteration: 86
Difference: 5.71%
Correct Classifications: 40171
Incorrect Classifications: 45829
Correct Rate: 46.71%
Incorrect Rate: 53.29%

Iteration: 1
Difference: 3.35%
Correct Classifications: 3
Incorrect Classifications: 997
Correct Rate: 0.30%
Incorrect Rate: 99.70%

Iteration: 6
Difference: 3.69%
Correct Classifications: 2421
Incorrect Classifications: 3579
Correct Rate: 40.35%
Incorrect Rate: 59.65%

Iteration: 11
Difference: 3.85%
Correct Classifications: 4825
Incorrect Classifications: 6175
Correct Rate: 43.86%
Incorrect Rate: 56.14%

Iteration: 16
Difference: 3.71%
Correct Classifications: 7218
Incorrect Classifications: 8782
Correct Rate: 45.11%
Incorrect Rate: 54.89%

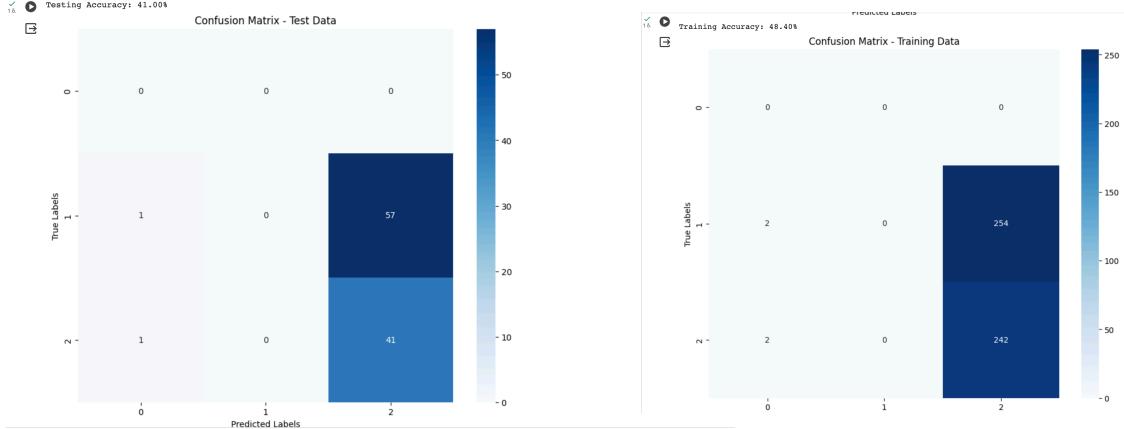
Final Correct Examples:
Input: [0.77254903 0.76862746 0.7529412 ... 0.02745098 0.01960784 0.02745098], True Label: [1], Predicted Label: 1
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1

Final Incorrect Examples:
Input: [0.58431375 0.7176471 0.81960785 ... 0.6431373 0.8156863 0.91764706], True Label: [0], Predicted Label: 1
Input: [0.8235294 0.8666667 0.8980392 ... 0.2509804 0.25882354 0.11764706], True Label: [0], Predicted Label: -1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.9333334 0.9529412], True Label: [0], Predicted Label: -1

Training Time: 6418.30 seconds

```

Number of Support Vectors: 244																	
Indices of Support Vectors: 1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17																	
18	19	28	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107
108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125
126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161
162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197
198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215
216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233
233	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251
252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269
276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293
288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305
306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323
324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341
342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359
360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377
378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395
396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413
414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431
432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449
450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467
468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485
486	487	488	489	490	491	492	493	494	495	496	497	498	499				



2η εκτέλεση :

Iteration: 86
 Difference: 6.85%
 Correct Classifications: 38934
 Incorrect Classifications: 47066
 Correct Rate: 45.27%
 Incorrect Rate: 54.73%

Iteration: 91
 Difference: 7.66%
 Correct Classifications: 40850
 Incorrect Classifications: 50150
 Correct Rate: 44.89%
 Incorrect Rate: 55.11%

Iteration: 96
 Difference: 7.98%
 Correct Classifications: 42835
 Incorrect Classifications: 53165
 Correct Rate: 44.62%
 Incorrect Rate: 55.38%

Iteration number exceeded the max of 100 iterations
 Final Correct Classifications: 44191
 Final Total Examples: 100000
 Final Accuracy: 44.19%

Final Correct Examples:

Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
 Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
 Input: [0.19067843 0.2509804 0.14509805 ... 0.6784314 0.64705884 0.63529414], True Label: [1], Predicted Label: 1

Final Incorrect Examples:

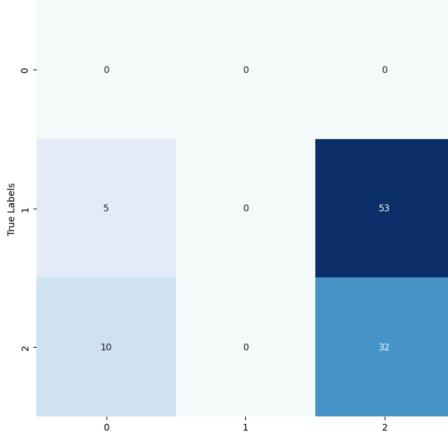
Input: [0.5566626 0.5921569 0.62352943 ... 0.52156866 0.5176471 0.49411765], True Label: [0], Predicted Label: 1
 Input: [0.87058824 0.8117647 0.7764706 ... 0.786803923 0.6431373], True Label: [0], Predicted Label: 1
 Input: [0.7921569 0.8 0.78039217 ... 0.941765 0.9333334 0.9529412], True Label: [0], Predicted Label: 1

Training Time: 6730.34 seconds

Decision Boundary: 996.28584018 575.05626248 inf inf inf 651.93419471
 Indices of Support Vectors: 244
 Number of Support Vectors: 244
 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
 486 487 488 489 490 491 492 493 494 495 496 497 498 499]

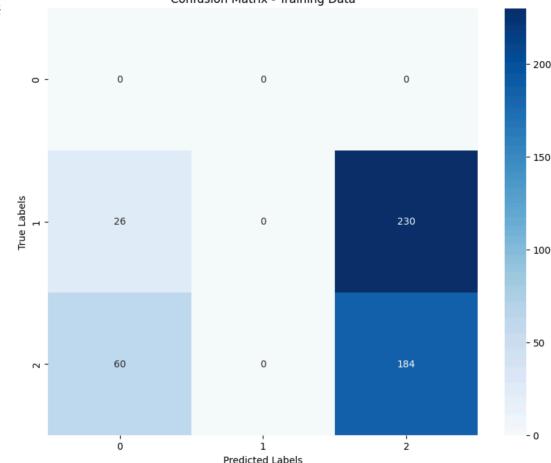
Testing Accuracy: 32.00%

Confusion Matrix - Test Data



Training Accuracy: 36.80%

Confusion Matrix - Training Data



3η εκτέλεση :

```
(200, 3072)
(200, 1)
(50, 3072)
(50, 1)
Iteration: 1
Difference: 2.08%
Correct Classifications: 212
Incorrect Classifications: 188
Correct Rate: 53.00%
Incorrect Rate: 47.00%

Iteration: 6
Difference: 2.57%
Correct Classifications: 1255
Incorrect Classifications: 1145
Correct Rate: 52.29%
Incorrect Rate: 47.71%

Iteration: 11
Difference: 1.95%
Correct Classifications: 2276
Incorrect Classifications: 2124
Correct Rate: 51.73%
Incorrect Rate: 48.27%

Iteration: 16
Difference: 2.14%
Correct Classifications: 3302
Incorrect Classifications: 3098
Correct Rate: 51.59%
Incorrect Rate: 48.41%

Iteration: 86
Difference: 2.11%
Correct Classifications: 17721
Incorrect Classifications: 16679
Correct Rate: 51.51%
Incorrect Rate: 48.49%

Iteration: 91
Difference: 2.47%
Correct Classifications: 18737
Incorrect Classifications: 17663
Correct Rate: 51.48%
Incorrect Rate: 48.52%

Iteration: 96
Difference: 2.36%
Correct Classifications: 19763
Incorrect Classifications: 18637
Correct Rate: 51.47%
Incorrect Rate: 48.53%

Iteration number exceeded the max of 100 iterations
Final Correct Classifications: 20599
Final Total Examples: 40000
Final Accuracy: 51.50%

Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255 ], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
Input: [0.02745098 0.02352941 0.00392157 ... 0.6078628 0.01960784 ], True Label: [1], Predicted Label: 1

Final Incorrect Examples:
Input: [0.64705884 0.68235296 0.7176471 ... 0.5176471 0.5647059 0.627451 ], True Label: [0], Predicted Label: 1
Input: [0.7294118 0.8117647 0.94509804 ... 0.7764707 0.78431374 0.8156863 ], True Label: [0], Predicted Label: 1
Input: [0.6039216 0.59607846 0.60784316 ... 0.33333334 0.3254902 0.3372549 ], True Label: [0], Predicted Label: 1
Training Time: 542.43 seconds

 boundary = model.decision_boundary()
print("Decision Boundary:", boundary)

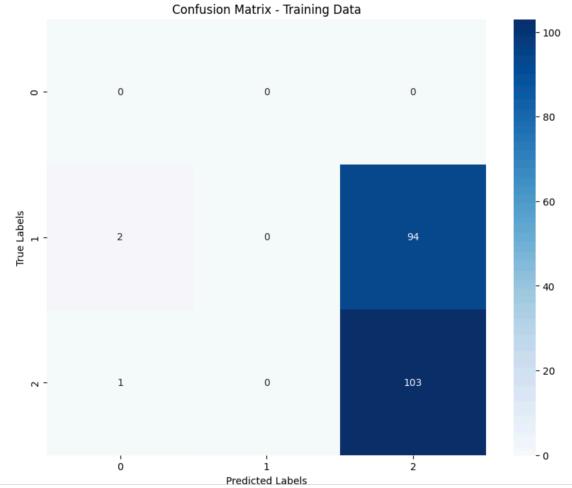
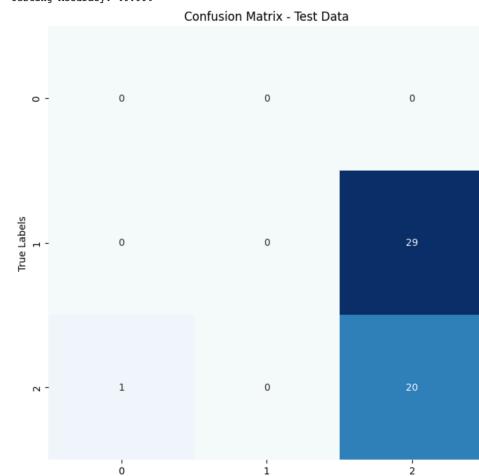
 Decision Boundary: [ 687.65612373 758.09080783 inf inf 907.74359561
inf 741.56952921 994.27612121 886.67104269 inf
581.24837473 457.33733091 843.03497155 561.72546185 601.93138369
inf 855.67391343 inf 470.80469747 811.84738684
518.09603689 899.0499804 816.61067799 728.51305538 inf
inf 587.77537442 468.08446599 inf 166.35698299
742.03879796 567.56781342 793.25968641 738.1631147 inf
885.7346327 598.54953914 inf 838.4529934 inf
inf inf 1648.3479511 581.17266797 893.26398022
inf inf 181.3221513 651.96561016 724.20792727
inf inf 181.3221513 651.96561016 724.20792727
709.2639446 1275.99490805 617.4858349 inf 601.72147367
inf inf 591.46958815 inf inf
591.21850877 677.76189773 678.02961047 inf 786.4598907
813.99460123 inf 725.04875781 866.43575982 716.49486075
inf inf inf inf inf
inf 750.43302681 1013.73667942 inf inf
600.67244577 inf 1231.88178638 inf 695.47668072
1214.17622107 inf inf inf inf
inf inf 1186.22043052 891.43723786 inf
inf inf inf 564.10927187 inf
inf 895.64299276 695.24863716 inf inf
inf 713.95745645 737.89427686 841.34596496 inf
713.26893238 inf 949.49667173 759.82910865 inf
942.49284781 613.81993221 inf inf 747.21738045
inf inf 1109.23245742 586.79981327 654.61140034 766.50606672
631.97470759 699.76854316 inf 710.45536073 inf
inf inf 530.12357182 inf inf
773.78105354 inf inf inf 578.95415442 inf
inf inf inf 647.42355848 inf
545.16833465 inf inf inf 904.47721918 inf
inf inf 717.39936117 inf inf
inf inf 409.64911587 994.48117197 inf
inf 463.58916768 inf 729.27966714 inf
1045.14422692 inf inf inf inf
675.58070854 inf 1151.23944243 1060.37266527 1269.02922947

 support_vectors = model.support_vectors
print("Number of Support Vectors:", support_vectors.shape[0])
print("Indices of Support Vectors:", np.where(np.isin(x_train_flattened, support_vectors).all(axis=1))[0])

 Number of Support Vectors: 104
Indices of Support Vectors: [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199]
```

Testing Accuracy: 40.00%

Training Accuracy: 51.50%



4η εκτέλεση :

```
(200, 3072)
(200, 1)
(50, 3072)
(50, 1)
Iteration: 1
Difference: 2.07%
Correct Classifications: 210
Incorrect Classifications: 190
Correct Rate: 52.50%
Incorrect Rate: 47.50%
```

```
Iteration: 6
Difference: 2.09%
Correct Classifications: 1256
Incorrect Classifications: 1144
Correct Rate: 52.33%
Incorrect Rate: 47.67%
```

```
Iteration: 11
Difference: 2.10%
Correct Classifications: 2302
Incorrect Classifications: 2098
Correct Rate: 52.32%
Incorrect Rate: 47.68%
```

```
Iteration: 16
Difference: 2.14%
Correct Classifications: 3323
Incorrect Classifications: 3077
Correct Rate: 51.92%
Incorrect Rate: 48.08%
```

```
Iteration: 196
Difference: 4.52%
Correct Classifications: 34672
Incorrect Classifications: 43728
Correct Rate: 44.22%
Incorrect Rate: 55.78%
```

```
Iteration number exceeded the max of 200 iterations
Final Correct Classifications: 35235
Final Total Examples: 80000
Final Accuracy: 44.04%
```

```
Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
Input: [1. 1. 1. ... 1. 1. 1.], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
```

```
Final Incorrect Examples:
Input: [0.89411765 0.95686275 0.9372549 ... 1. 1. 1. ], True Label: [0], Predicted Label: 1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.9333334 0.9529412 ], True Label: [0], Predicted Label: 1
Input: [0.49411765 0.4627451 0.43137255 ... 0.4745098 0.44313726 0.4 ], True Label: [0], Predicted Label: 1
Training Time: 1125.97 seconds
```

```
Iteration: 66
Difference: 2.25%
Correct Classifications: 13604
Incorrect Classifications: 12796
Correct Rate: 51.53%
Incorrect Rate: 48.47%
```

```
Iteration: 71
Difference: 1.93%
Correct Classifications: 14627
Incorrect Classifications: 13773
Correct Rate: 51.50%
Incorrect Rate: 48.50%
```

```
Iteration: 76
Difference: 2.96%
Correct Classifications: 15655
Incorrect Classifications: 14745
Correct Rate: 51.50%
Incorrect Rate: 48.50%
```

```
Iteration: 81
Difference: 4.52%
Correct Classifications: 16602
Incorrect Classifications: 15798
Correct Rate: 51.24%
Incorrect Rate: 48.76%
```

```
Iteration: 136
Difference: 5.64%
Correct Classifications: 26517
Incorrect Classifications: 27883
Correct Rate: 48.74%
Incorrect Rate: 51.26%
```

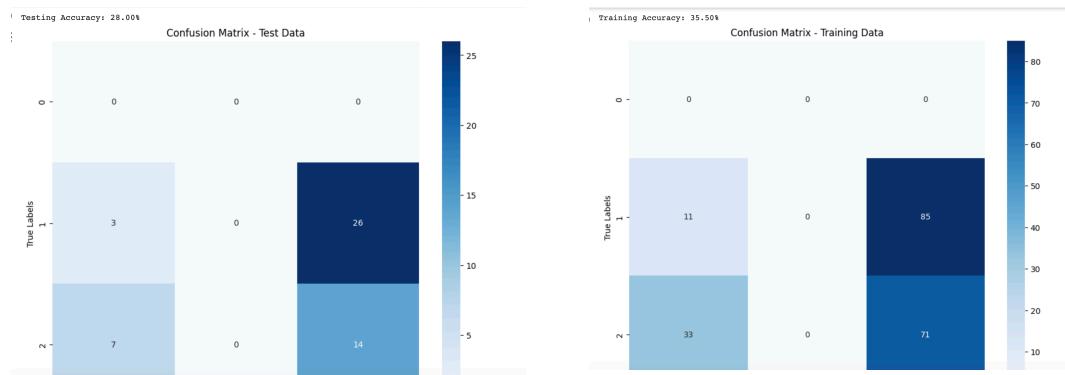
```
Iteration: 141
Difference: 5.32%
Correct Classifications: 27225
Incorrect Classifications: 29175
Correct Rate: 48.27%
Incorrect Rate: 51.73%
```

```
Iteration: 146
Difference: 5.08%
Correct Classifications: 27902
Incorrect Classifications: 30498
Correct Rate: 47.78%
Incorrect Rate: 52.22%
```

```
Iteration: 151
Difference: 5.74%
Correct Classifications: 28545
Incorrect Classifications: 31855
Correct Rate: 47.26%
Incorrect Rate: 52.74%
```

```
Decision Boundary: [1.06326685e+03 3.46960964e+02 inf inf
9.48976989e+03 inf 1.87985804e+03 3.66829036e+02
2.13752321e+03 1.37425774e+03 1.49159165e+03 inf
1.63042848e+03 2.38512595e+03 3.67711385e+02
3.36250721e+03 2.80120330e+03 2.31601886e+03
inf inf 1.16550998e+03 8.64794172e+02
inf 2.16067776e+02 3.64044365e+02 7.60642148e+02
2.5650762e+03 1.64045893e+02 inf 2.49716657e+02
6.79229762e+03 inf 2.72712539e+02 inf
inf inf 2.73641688e+02 1.52509083e+03
1.93688585e+03 inf
1.9691819e+03 8.7327224e+02 inf 2.21810954e+03
9.9752490e+02 1.31393475e+03 2.29967495e+03 2.70534831e+02
4.61357750e+02 2.13696758e+03 inf 2.7017182e+02
inf inf 2.76165296e+03 inf
inf 8.92488820e+02 3.11451576e+03 1.12266326e+04
inf 2.74619218e+03 1.28045629e+03 inf
1.33948443e+03 3.2224916e+02 2.08895638e+03 inf
inf inf inf inf
inf 3.69189769e+02 3.26595562e+03
inf 2.63989448e+02 inf 4.89787368e+02
inf 1.46507927e+02 2.52826284e+02 inf
inf 4.59743178e+02 4.31237828e+02 inf
inf inf inf 1.08278265e+19
inf inf inf 2.38156042e+03 inf
1.62863611e+03 1.61850328e+03 4.68741834e+02
inf 2.40932118e+02 inf inf
3.13647251e+03 inf inf inf]
```

```
Number of Support Vectors: 103
Indices of Support Vectors: [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 ]
```



5η εκτέλεση :

```
(200, 3072)  
(200, 1)  
(50, 3072)  
(50, 1)  
Iteration: 1  
Difference: 2.24%  
Correct Classifications: 3  
Incorrect Classifications: 397  
Correct Rate: 0.75%  
Incorrect Rate: 99.25%
```

```
Iteration: 6  
Difference: 2.29%  
Correct Classifications: 573  
Incorrect Classifications: 1827  
Correct Rate: 23.88%  
Incorrect Rate: 76.12%
```

```
Iteration: 11  
Difference: 2.63%  
Correct Classifications: 1606  
Incorrect Classifications: 2794  
Correct Rate: 36.50%  
Incorrect Rate: 63.50%
```

```
Iteration: 16  
Difference: 2.15%  
Correct Classifications: 2651  
Incorrect Classifications: 3749  
Correct Rate: 41.42%  
Incorrect Rate: 58.58%
```

```
Iteration: 46  
Difference: 2.56%  
Correct Classifications: 8838  
Incorrect Classifications: 9562  
Correct Rate: 48.03%  
Incorrect Rate: 51.97%
```

```
Iteration number exceeded the max of 50 iterations  
Final Correct Classifications: 9673  
Final Total Examples: 20000  
Final Accuracy: 48.37%
```

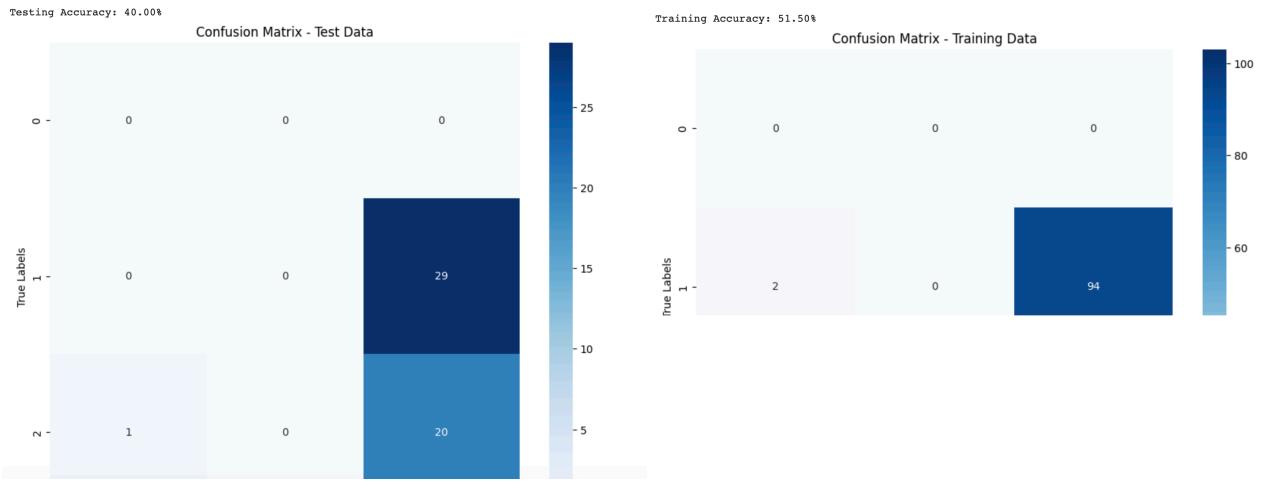
Final Correct Examples:
Input: [0.2627451 0.34059805 0.19215687 ... 0.7607843 0.8392157 0.36078432], True Label: [1], Predicted Label: 1
Input: [0.6666667 0.7058824 0.7764760 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
Input: [0.6335394 0.4 ... 0.3967844 0.7327255 0.23253041 0.072459891], True Label: [1], Predicted Label: 1

```

Final Incorrect Examples:
Input: [0.6313726 0.68235296 0.72156864 ... 0.3137255 0.35686275 0.09019608], True Label: [0], Predicted Label: 1
Input: [0.4862745 0.5137255 0.64705884 ... 0.5176471 0.49803922 0.48235294], True Label: [1], Predicted Label: -1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.93333334 0.9529412 ], True Label: [0], Predicted Label: -1
Training Time: 276.45 seconds

```

Decision Boundary	1	793_45112937	774_47880418	inf	inf	inf	inf	inf
	inf	542_18935211	490_46356529	796_57550804	398_18295895			
662, 78385821	842_21054756	654_67327843	754_81111381	398_18295895				
	inf	851_73086507	inf	499_20582383	1899_8547272			
799, 20579859	654_67327843	1005_67451139	479_75986216	189_05131864				
	inf	696_59388441	654_65713181	690_93750818	691_67964496	inf		
1810, 20427145	677_2485938	inf	863_51436931	1234_92756777	738_48808030			
	inf	1099_464111	inf	597_55462826	447_15132389			
608, 10946411	543_9742472	539_48452194	495_44086037	665_74284542				
	inf	574_20933282	578_76179479	inf	918_298805			
1006, 29748955	inf	846_26847879	inf	891_89218162				
1006, 54694256	inf	633_2323678	615_71086826	693_55154121				
	inf	inf	inf	inf	inf			
621, 53941347	inf	873_89689941	926_74941505	inf	inf			
	inf	inf	inf	inf	inf			
854_55324207	inf	923_31395642	inf	785_74887915				
	inf	inf	inf	inf	inf			
inf	inf	1102_94683246	661_64679759	inf	inf			
inf	inf	inf	792_71719882	inf	inf			
inf	inf	1244_19586777	651_85469121	inf	inf			
inf	inf	555_60825417	973_64598332	695_67688514	inf			
791_48662354	inf	inf	834_38646475	631_98387575	inf			
	inf	inf	inf	inf	inf			
791_62235745	inf	973_76162585	785_18154362	592_53756687				
	inf	inf	inf	inf	inf			
549, 95319572	746_6442757	744_8842826	812_45158833	inf	inf			
	inf	inf	inf	inf	inf			
643_87221542	inf	inf	619_04277693	inf	inf			
	inf	inf	inf	inf	inf			
517_38826248	inf	inf	inf	896_85227843				
	inf	inf	inf	inf	inf			
inf	inf	inf	inf	inf	inf			
inf	inf	541_74495183	inf	743_4228548	inf			
666_54524745	inf	inf	inf	inf	inf			
991_42623573	inf	inf	inf	inf	inf			
839_59375123	508_7398522	829_9885717	inf	897_41641121				



6η εκτέλεση :

```
(200, 3072)
(200, 1)
(50, 3072)
(50, 1)
Iteration: 1
Difference: 2.24%
Correct Classifications: 6
Incorrect Classifications: 394
Correct Rate: 1.50%
Incorrect Rate: 98.50%

Iteration: 6
Difference: 3.90%
Correct Classifications: 651
Incorrect Classifications: 1749
Correct Rate: 27.12%
Incorrect Rate: 72.88%

Iteration: 11
Difference: 5.06%
Correct Classifications: 1287
Incorrect Classifications: 3113
Correct Rate: 29.25%
Incorrect Rate: 70.75%


Iteration number exceeded the max of 100 iterations
Final Correct Classifications: 14359
Final Total Examples: 40000
Final Accuracy: 35.90%

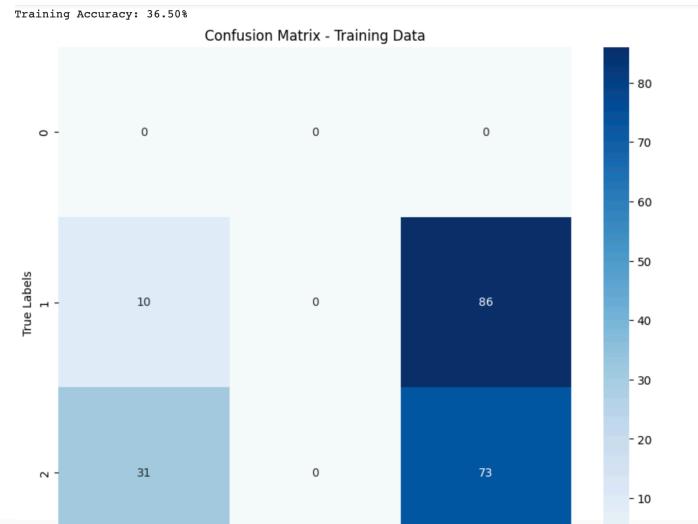
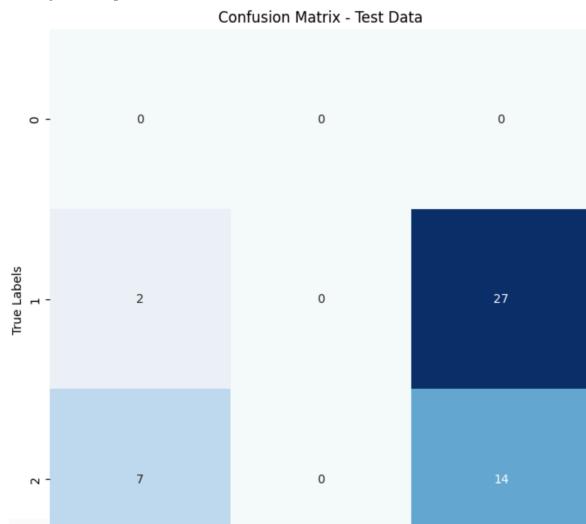
Final Correct Examples:
Input: [1. 1. 1. ... 1. 1. 1.], True Label: [1], Predicted Label: 1
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255 ], True Label:
Input: [0.24313726 0.2509804 0.17254902 ... 0.6627451 0.6117647 0.627451 ], True Label:

Final Incorrect Examples:
Input: [0.72156864 0.7764706 0.827451 ... 0.68235296 0.7137255 0.75686276], True Label:
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.93333334 0.9529412 ], True Label:
Input: [0.98039216 0.98039216 0.98039216 ... 0.98039216 0.98039216 0.98039216], True Label:
Training Time: 568.16 seconds
```

```
print("Number of Support Vectors:", support_vectors.shape[0])
print("Indices of Support Vectors:", np.where(np.isin(x_train_flattened, support_vectors).all(axis=1))[0])
```

```
Decision Boundary: [1.03845002e+03 2.66661364e+02 inf
3.89542357e+02 inf 1.15616299e+03 2.56502816e+02
1.41010809e+04 inf 6.03203434e+03 1.3618230e+04
8.01871066e+03 6.78722167e+03 2.16672520e+03 inf
2.42479332e+03 inf 5.66326106e+03 2.51622680e+02
2.43590525e+02 2.71065287e+02 4.23455697e+03 7.50723738e+03
inf inf 3.93464518e+03 3.08218111e+03
inf 3.11316389e+02 2.43590525e+02 2.25189159e+03
1.80197260e+03 1.40525129e+03 inf 2.43972247e+02
4.38813679e+18 inf 2.43590525e+02 inf
inf inf 2.49136657e+02 2.14154397e+03
1.46615236e+03 inf inf inf
7.88482672e+03 3.40127804e+03 inf 1.83846210e+03
1.39124668e+04 1.11421280e+03 1.78649757e+03 2.43590525e+02
3.00865070e+02 3.63176275e+03 inf 2.43590525e+02
inf inf 1.94380451e+03 inf
inf 6.64251306e+02 1.14325486e+03 1.71095454e+03
inf 2.92391918e+03 1.29287297e+03 inf
5.16899169e+03 2.49097647e+02 1.45758889e+03 inf
inf inf inf 1.75525472e+19
inf 2.49923278e+02 1.61223222e+04 inf
inf 5.16122372e+03 inf 3.61481373e+02
inf 9.21372257e+03 2.47205056e+02 inf
inf inf inf inf
inf 3.41745609e+02 2.84826061e+02 inf
inf inf inf 1.95245151e+03
inf inf inf 4.87950594e+03 2.45614766e+03
inf inf inf 5.59412455e+03
3.01575606e+03 5.52961757e+03 inf 7.59839816e+04
```

Testing Accuracy: 28.00%



7η εκτέλεση :

(200, 3072)
(200, 1)
(50, 3072)
(50, 1)
Iteration: 1
Difference: 2.35%
Correct Classifications: 209
Incorrect Classifications: 191
Correct Rate: 52.25%
Incorrect Rate: 47.75%

Iteration: 6
Difference: 5.11%
Correct Classifications: 691
Incorrect Classifications: 1709
Correct Rate: 28.79%
Incorrect Rate: 71.21%

Iteration: 11
Difference: 5.61%
Correct Classifications: 1225
Incorrect Classifications: 3175
Correct Rate: 27.84%
Incorrect Rate: 72.16%

Iteration: 16
Difference: 2.43%
Correct Classifications: 1894
Incorrect Classifications: 4506
Correct Rate: 29.59%
Incorrect Rate: 70.41%

Iteration: 91
Difference: 2.15%
Correct Classifications: 17258
Incorrect Classifications: 19142
Correct Rate: 47.41%
Incorrect Rate: 52.59%

Iteration: 96
Difference: 2.00%
Correct Classifications: 18297
Incorrect Classifications: 20103
Correct Rate: 47.65%
Incorrect Rate: 52.35%

Iteration number exceeded the max of 100 iterations
Final Correct Classifications: 19123
Final Total Examples: 40000
Final Accuracy: 47.81%

Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
Input: [0.08627451 0.09803922 0.10980392 ... 0.69883923 0.7019608 0.68235296], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1

Final Incorrect Examples:
Input: [0.27058825 0.5294118 0.73333335 ... 0.5294118 0.73333335 0.8784314], True Label: [0], Predicted Label: 1
Input: [0.24313726 0.24313726 0.24313726 ... 1. 1. 1. 1.], True Label: [0], Predicted Label: 1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.93333334 0.9529412], True Label: [0], Predicted Label: 1
Training Time: 563.24 seconds

Iteration: 41
Difference: 3.09%
Correct Classifications: 6971
Incorrect Classifications: 9429
Correct Rate: 42.51%
Incorrect Rate: 57.49%

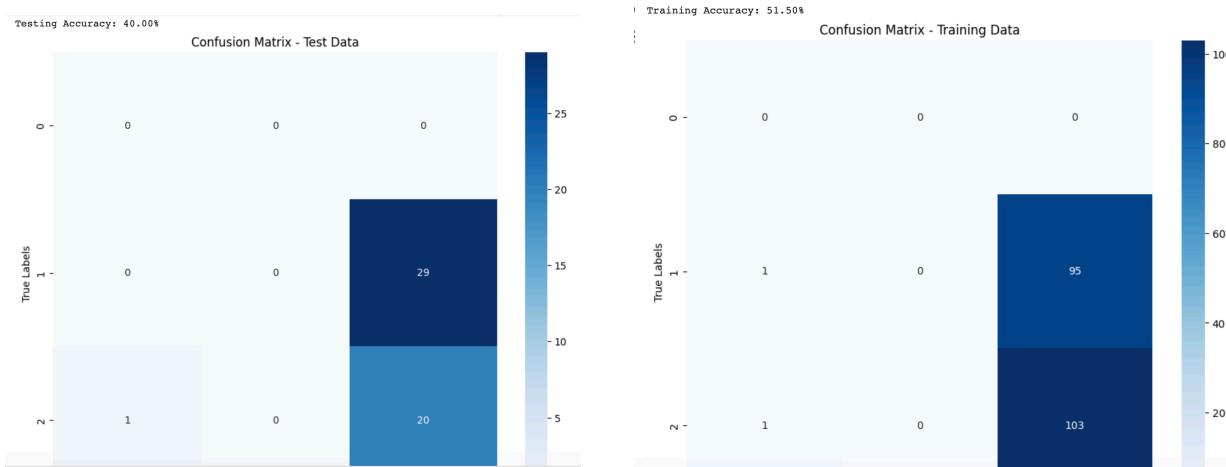
Iteration: 46
Difference: 2.19%
Correct Classifications: 7989
Incorrect Classifications: 10411
Correct Rate: 43.42%
Incorrect Rate: 56.58%

Iteration: 51
Difference: 2.08%
Correct Classifications: 9021
Incorrect Classifications: 11379
Correct Rate: 44.22%
Incorrect Rate: 55.78%

Iteration: 56
Difference: 2.57%
Correct Classifications: 10057
Incorrect Classifications: 12343
Correct Rate: 44.90%
Incorrect Rate: 55.10%

Decision Boundary: [6.78848721e+02 7.07707456e+02 inf inf
9.19311619e+02 inf 6.70349736e+02 7.14574058e+02
7.4838026e+02 inf 8.25128350e+02 6.16303366e+02
8.52211977e+02 6.50985885e+02 4.61072830e+02 inf
8.52311474e+02 inf 7.73324421e+02 7.45312505e+02
5.92553579e+02 9.48683908e+02 1.07130130e+03 7.63135925e+02
inf inf 9.07834680e+02 5.67487628e+02
inf 1.95493281e+02 6.74888211e+02 6.58276241e+02
8.86884062e+02 9.00524581e+02 inf 8.98231205e+02
4.79993518e+02 inf 8.85295051e+02 inf
inf 2.25388407e+20 7.12402056e+02 1.02336344e+03
9.84102315e+02 inf inf inf
1.10633994e+03 6.71286160e+02 inf 7.38041234e+02
6.73492537e+02 4.25465615e+02 6.34315522e+02 6.87785538e+02
8.77129203e+02 8.77786897e+02 inf 6.43036541e+02
inf inf 4.61854945e+02 inf
inf 6.63113985e+02 7.95689310e+02 7.33623795e+02
inf 7.488807705e+02 7.30777318e+02 inf
8.00299649e+02 7.53060726e+02 6.19304481e+02 inf
inf inf inf inf
inf 7.17060384e+02 1.34626378e+03 inf
inf 6.27976275e+02 inf 6.98064189e+02 inf
inf 6.137772909e+02 7.66694514e+02 inf
inf inf inf inf
inf 8.65949590e+02 7.67415589e+02 inf
inf inf inf 5.17335196e+02
2.25388407e+20 inf 1.29599948e+03 5.97609987e+02
inf inf inf 7.20885285e+02
6.65187858e+02 5.89624436e+02 inf 8.61389216e+02
inf 6.23141412e+02 6.02633104e+02 inf
8.46656818e+02 7.66747281e+02 inf inf
9.05955035e+02 inf 1.10344726e+03 7.27977410e+02
6.25035303e+02 6.99051998e+02 6.84684639e+02 6.78796824e+02
inf 5.907336460e+02 inf inf
inf 4.94839618e+02 inf inf
8.017411779e+02 inf inf inf
6.91465404e+02 inf inf inf

Number of Support Vectors: 106
Indices of Support Vectors: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106
107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199]



8η εκτέλεση :

```

Testing Accuracy: 40.00%
Confusion Matrix - Test Data
True Labels
0 - 0 0 0 29 0
1 - 0 0 20 0 29
2 - 1 0 20 103 0

Training Accuracy: 51.50%
Confusion Matrix - Training Data
True Labels
0 - 0 0 0 0
1 - 1 0 0 95
2 - 1 0 0 103

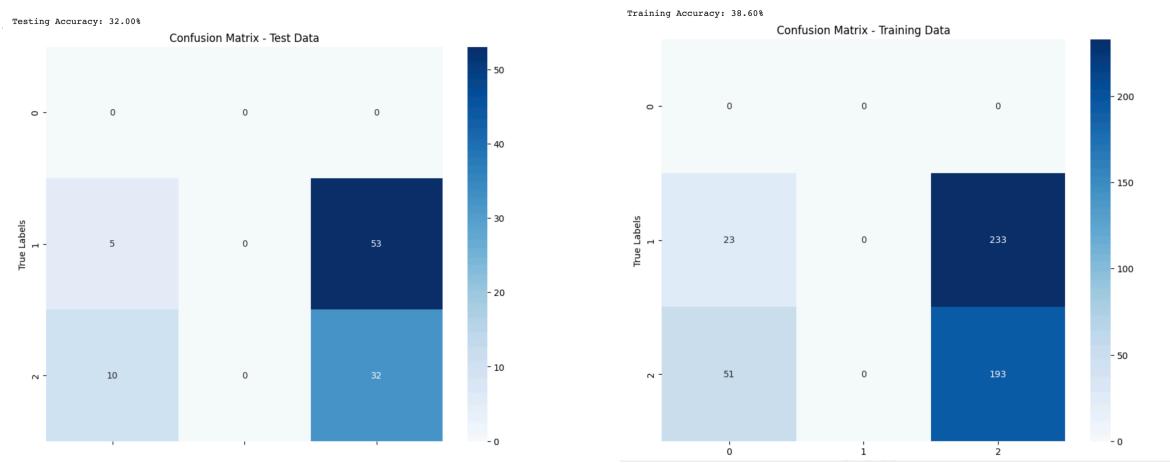
(500, 3072)
(500, 1)
(100, 3072)
(100, 1)
Iteration: 1
Difference: 3.61%
Correct Classifications: 500
Incorrect Classifications: 500
Correct Rate: 50.00%
Incorrect Rate: 50.00%
Iteration: 41
Difference: 4.49%
Correct Classifications: 15814
Incorrect Classifications: 25186
Correct Rate: 38.57%
Incorrect Rate: 61.43%
Iteration: 46
Difference: 6.16%
Correct Classifications: 17899
Incorrect Classifications: 28101
Correct Rate: 38.91%
Incorrect Rate: 61.09%
Iteration: 51
Difference: 8.67%
Correct Classifications: 19614
Incorrect Classifications: 31386
Correct Rate: 38.46%
Incorrect Rate: 61.54%
Iteration: 56
Difference: 6.41%
Correct Classifications: 21627
Incorrect Classifications: 34373
Correct Rate: 38.62%
Incorrect Rate: 61.38%
Iteration: 61
Difference: 7.84%
Correct Classifications: 23662
Incorrect Classifications: 37338
Correct Rate: 38.79%
Incorrect Rate: 61.21%
Iteration: 86
Difference: 7.05%
Correct Classifications: 32838
Incorrect Classifications: 53162
Correct Rate: 38.18%
Incorrect Rate: 61.82%
Iteration: 91
Difference: 8.26%
Correct Classifications: 34552
Incorrect Classifications: 56448
Correct Rate: 37.97%
Incorrect Rate: 62.03%
Iteration: 96
Difference: 7.29%
Correct Classifications: 36290
Incorrect Classifications: 59710
Correct Rate: 37.88%
Incorrect Rate: 62.12%
Iteration number exceeded the max of 100 iterations
Final Correct Classifications: 37838
Final Total Examples: 100000
Final Accuracy: 37.84%
Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019608 0.3137255], True Label: [1], Predicted Label: 1
Input: [1. 1. 1. ... 0.6509804 0.6509804 0.6509804], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
Input: [0.47843137 0.6039216 0.7921569 ... 0.98839216 0.9882353], True Label: [0], Predicted Label: 1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.93333334 0.9529412], True Label: [0], Predicted Label: 1
Training Time: 6839.14 seconds

```

```

Number of Support Vectors: 246
Indices of Support Vectors: [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
486 487 488 489 490 491 492 493 494 495 496 497 498 499

```



9η εκτέλεση :

```
model = SVM()
model.set_params(max_iter=1000, kernel_type='polynomial', C=1.0, epsilon=0.001, degree=2)

start_time = time.time()

model.fit(x_train_flattened, y_train_subset)

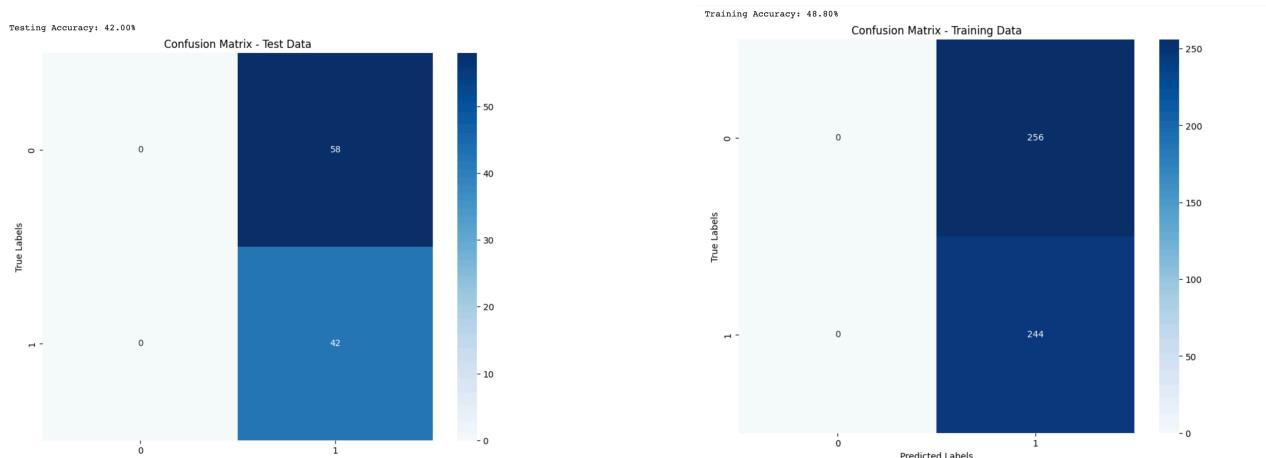
training_time = time.time() - start_time
print("Training Time: {:.2f} seconds".format(training_time))

train_predictions = model.predict(x_train_flattened)

(500, 3072)
(500, 1)
(100, 3072)
(100, 1)
Training finished after 1 iterations
Final Correct Classifications: 469
Final Total Examples: 1000
Final Accuracy: 46.90%
Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.30196808 0.31372551], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
Input: [0.46666667 0.5254982 0.6509884 ... 0.48235294 0.5411765 0.6627451], True Label: [1], Predicted Label: 1

Final Incorrect Examples:
Input: [0.22352941 0.43529412 0.5764786 ... 0.18431373 0.45882353 0.59687846], True Label: [0], Predicted Label: 1
Input: [0.15686275 0.34901962 0.627451 ... 0.75868276 0.8117647 0.9254902], True Label: [0], Predicted Label: 1
Input: [0.7921363 0.8 0.78039217 ... 0.9411765 0.9333334 0.9529412], True Label: [0], Predicted Label: 1
Training Time: 04.91 seconds

Number of Support Vectors: 125
Indices of Support Vectors: [ 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17
 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99  100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
486 487 488 489 490 491 492 493 494 495 496 497 498 499]
```



10η εκτέλεση :

```

model = SVM()
model.set_params(max_iter=1000, kernel_type='polynomial', C=1.0, epsilon=0.001, degree=2)

start_time = time.time()

model.fit(x_train_flattened, y_train_subset)

training_time = time.time() - start_time
print("Training Time: {:.2f} seconds".format(training_time))

train_predictions = model.predict(x_train_flattened)

(500, 3072)
(100, 1)
(300, 3072)
(100, 1)
Training finished after 1 iterations
Number of Correct Classifications: 469
Final Total Examples: 1000
Final Accuracy: 46.9%
Final Correct Examples:
Input: [0.66666667 0.5985824 0.5674706 ... 0.28627452 0.3019608 0.3137255 ...], True Label: [1], Predicted Label: 1
Input: [0.62352941 0.4 0.39687844 ... 0.7137255 0.22352941 0.07459891 ...], True Label: [1], Predicted Label: 1
Input: [0.46666667 0.5254902 0.6589884 ... 0.48235294 0.5411765 0.6627451 ...], True Label: [1], Predicted Label: 1

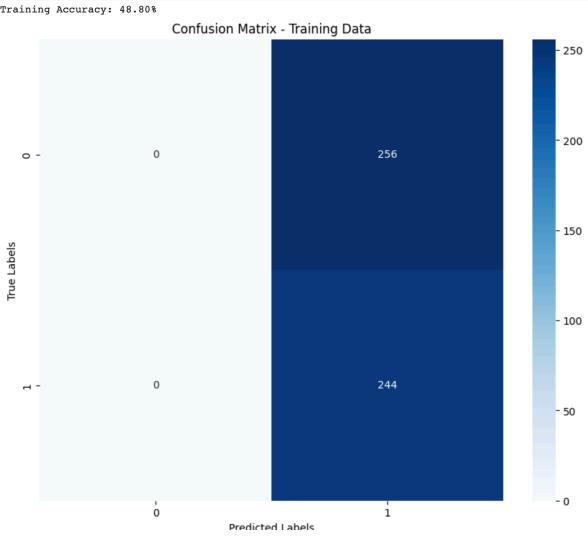
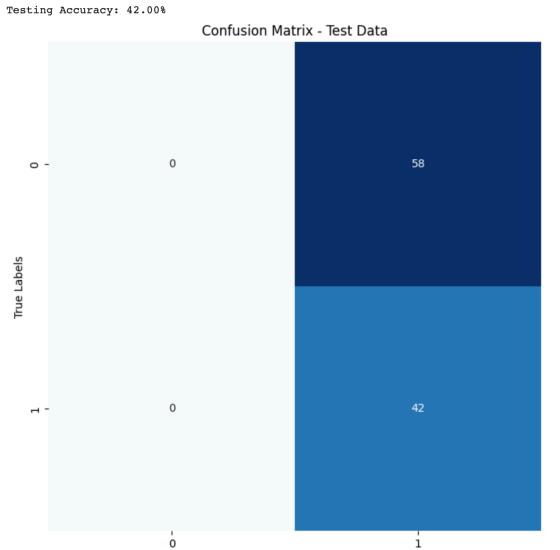
Final Incorrect Examples:
Input: [0.22352941 0.43529412 0.5764706 ... 0.18431373 0.45882353 0.59687846 ...], True Label: [0], Predicted Label: 1
Input: [0.15686275 0.34981962 0.627451 ... 0.75686276 0.8117647 0.9254982 ...], True Label: [0], Predicted Label: 1
Input: [0.79215987 0.8 0.780839217 ... 0.9411765 0.95333334 0.9529412 ...], True Label: [0], Predicted Label: 1
Training Time: 64.91 Seconds

Number of Support Vectors: 125
Indices of Support Vectors: [ 0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99  100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
486 487 488 489 490 491 492 493 494 495 496 497 498 499]

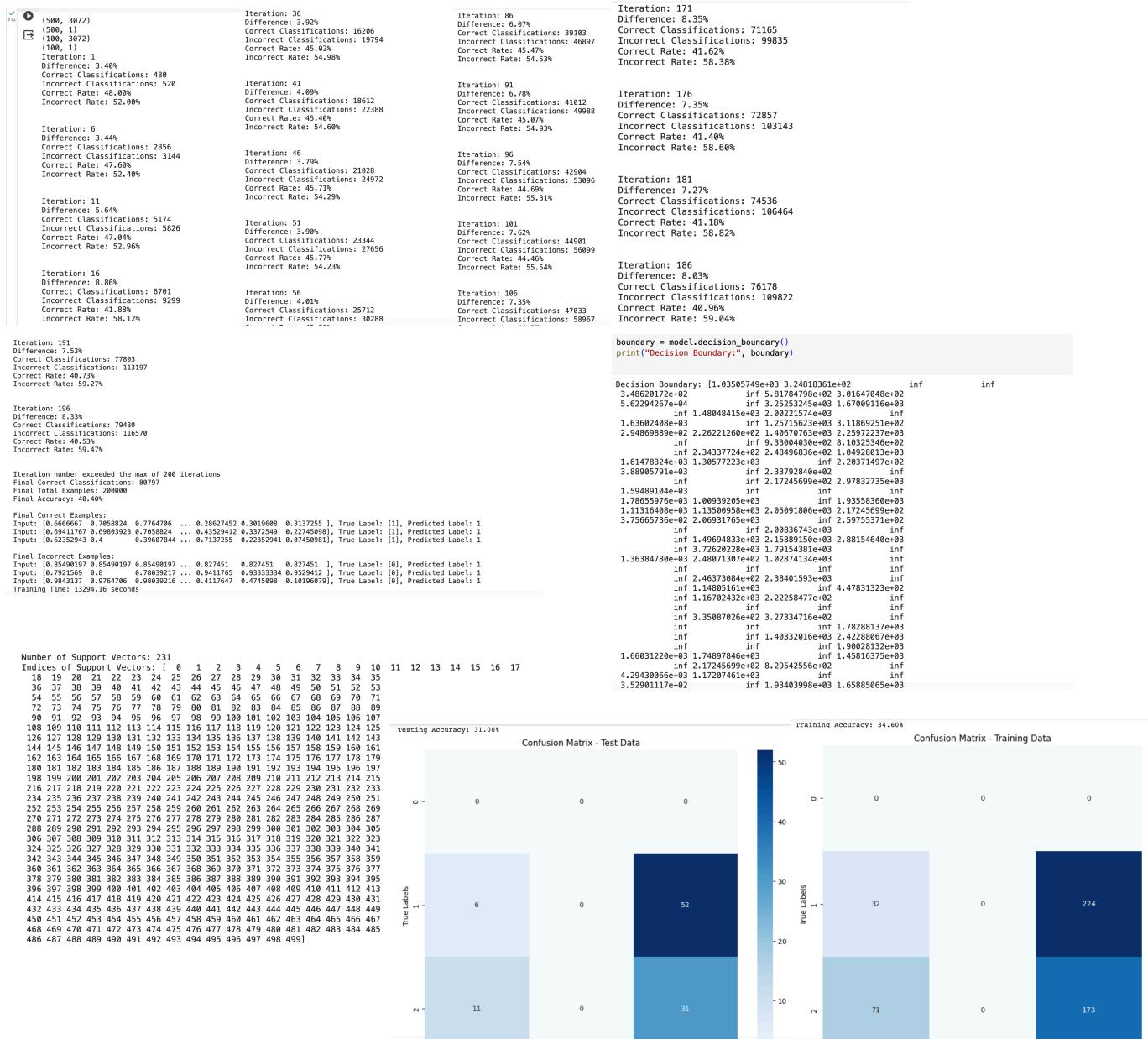
```

```
boundary = model.decision_boundary()
print("Decision Boundary:", boundary)
```

-inf -inf -inf -inf
-inf -inf -inf -inf
-inf -inf -inf -4.16304551e+09
-inf -inf -9.31861864e+08 -9.64893230e+08
-inf -inf -inf -inf
-inf -4.88530108e+10 -2.4533372e+08 -inf
-4.74976009e+08 -inf -inf -inf
-inf -inf -inf -inf
-inf -6.22149217e+09 -5.83865355e+08
-inf -inf -inf -inf
-inf -inf -1.35752767e+09 -inf
-6.53722469e+09 -inf -inf -inf
-1.23330308e+09 -6.01344948e+09 -inf -2.47678480e+09
-1.58958923e+09 -inf -inf -inf
-inf -inf -2.67580219e+09 -inf
-1.42403725e+09 -inf -inf -5.35243629e+09
-inf -inf -inf -2.55018743e+09
-inf -7.10977050e+09 -inf
-inf -inf -inf -inf
-inf -inf -inf -inf
-inf -inf -inf -inf
-1.60821374e+09 -2.35462957e+09 -8.10843977e+09 -inf
-6.25603907e+09 -inf -inf -inf
-6.04233191e+09 -inf -inf -inf
-inf -5.03273868e+09 -inf -inf
-5.61301771e+09 -inf -inf -inf
-inf -inf -2.33388229e+09 -inf
-inf -inf -inf -inf
-inf -inf -inf -2.44296004e+08
-inf -inf -1.76000176e+09 -inf
-inf -5.64638660e+08 -inf -inf
-inf -inf -5.34549355e+08 -inf
-inf -inf -inf -5.90275898e+09
-inf -2.28118649e+09 -2.35482277e+08 -3.68250470e+08
-1.inf -inf -inf -7.04552538e+08
-inf -1.42777251e+09 -3.53866773e+09 -7.06452586e+08
-inf -inf -inf -inf



11η εκτέλεση :



12η εκτέλεση :

```
(500, 3072)
(500, 1)
(100, 3072)
(100, 1)
Iteration: 1
Difference: 3.43%
Correct Classifications: 466
Incorrect Classifications: 534
Correct Rate: 46.60%
Incorrect Rate: 53.40%
Iteration: 41
Difference: 3.67%
Correct Classifications: 19687
Incorrect Classifications: 21313
Correct Rate: 48.02%
Incorrect Rate: 51.98%
Iteration: 46
Difference: 3.63%
Correct Classifications: 22073
Incorrect Classifications: 23927
Correct Rate: 47.98%
Incorrect Rate: 52.02%
Iteration: 51
Difference: 3.23%
Correct Classifications: 24499
Incorrect Classifications: 26501
Correct Rate: 48.04%
Incorrect Rate: 51.96%
Iteration: 56
Difference: 3.80%
Correct Classifications: 26948
Incorrect Classifications: 29052
Correct Rate: 48.12%
Incorrect Rate: 51.88%
Iteration: 61
Difference: 3.51%
Correct Classifications: 29348
Incorrect Classifications: 31652
Correct Rate: 48.11%
Iteration: 91
Difference: 4.26%
Correct Classifications: 43717
Incorrect Classifications: 47283
Correct Rate: 48.04%
Incorrect Rate: 51.96%
Iteration: 96
Difference: 4.26%
Correct Classifications: 46026
Incorrect Classifications: 49974
Correct Rate: 47.94%
Incorrect Rate: 52.06%
Iteration number exceeded the max of 100 iterations
Final Correct Classifications: 47831
Final Total Examples: 100000
Final Accuracy: 47.83%
Final Correct Examples:
Input: [0.6666667 0.7058824 0.7764706 ... 0.28627452 0.3019688 0.3137255], True Label: [1], Predicted Label: 1
Input: [0.30588236 0.44705883 0.5863922 ... 0.92156863 0.9019688 0.8627451], True Label: [1], Predicted Label: 1
Input: [0.62352943 0.4 0.39607844 ... 0.7137255 0.22352941 0.07450981], True Label: [1], Predicted Label: 1
Final Incorrect Examples:
Input: [0.6666667 0.6313726 0.7019688 ... 0.08627451 0.08235904 0.2], True Label: [0], Predicted Label: 1
Input: [0.23137255 0.16470589 0.12156863 ... 0.6156863 0.4392157 0.2784314], True Label: [0], Predicted Label: 1
Input: [0.7921569 0.8 0.78039217 ... 0.9411765 0.9333334 0.9529412], True Label: [0], Predicted Label: 1
Training Time: 6636.54 seconds
```

```
boundary = model.decision_boundary()
print("Decision Boundary:", boundary)
```

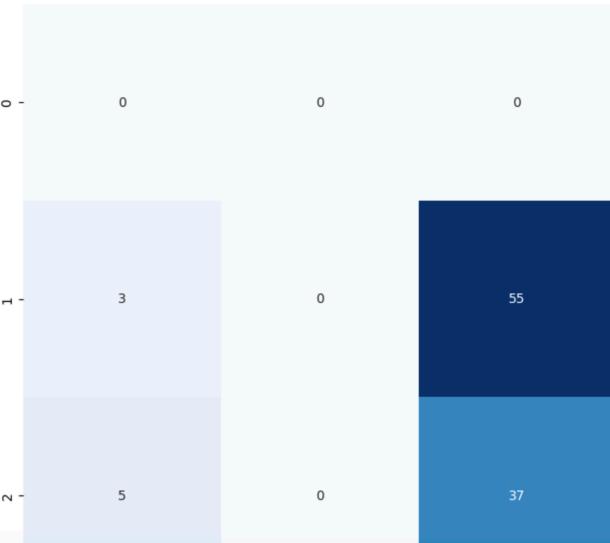
```
inf 719.90154279 1034.40040643 486.32344112 608.82234477
inf inf 936.29874721 816.28862358 928.6166403
308.31628027 inf 785.58302529 531.5787632 inf
774.09514711 inf inf inf inf
inf 711.52774169 inf 917.41098995 inf
inf 683.33322404 inf inf inf
inf inf inf inf inf
inf inf 676.78420312 653.20840054
inf inf 742.40384337 inf inf
inf inf 536.47953115 1012.7726756 inf
546.51589741 718.49919149 650.82129024 inf inf
581.39316465 inf inf inf inf
868.43850593 710.97986001 inf 756.80932159 547.64898282
inf inf 530.65089168 909.95465591 inf
865.15489387 572.48198606 694.82822003 671.85737813 inf
inf inf inf 1052.37770522 629.97579161
inf 995.35576996 inf 520.55881142 inf
inf inf 765.89615488 inf inf
inf 726.52968913 579.0856834 inf inf
1038.76609355 inf inf inf 767.9736904
397.98746583 921.1151127 918.91026535 672.549986 inf
523.69115682 559.57537963 838.77298556 773.6892232 831.62357288
537.51172263 inf inf inf inf
598.84066718 inf 604.81710256 726.23934334 inf
701.18271469 587.460393 inf inf 775.63427074
inf 628.06606505 792.55850139 inf inf
inf 840.01858585 inf 455.00187661 732.00334213
inf 498.21577372 inf inf inf
inf 945.79107259 863.97841854 620.75713423 inf
735.12062237 inf inf inf 673.845047
133.16870333 743.7171353 inf inf 691.17995468
inf inf 863.54590093 inf inf
```

```
support_vectors = model.support_vectors
print("Number of Support Vectors:", support_vectors.shape[0])
print("Indices of Support Vectors:", np.where(np.isin(x_train_flattened, support_vectors).all(axis=1))[0])

Number of Support Vectors: 244
Indices of Support Vectors: [ 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17
 18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35
 36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53
 54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99  100 101 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215
216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233
234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251
252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269
270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287
288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305
306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341
342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359
360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377
378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395
396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413
414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431
432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467
468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
486 487 488 489 490 491 492 493 494 495 496 497 498 499]
```

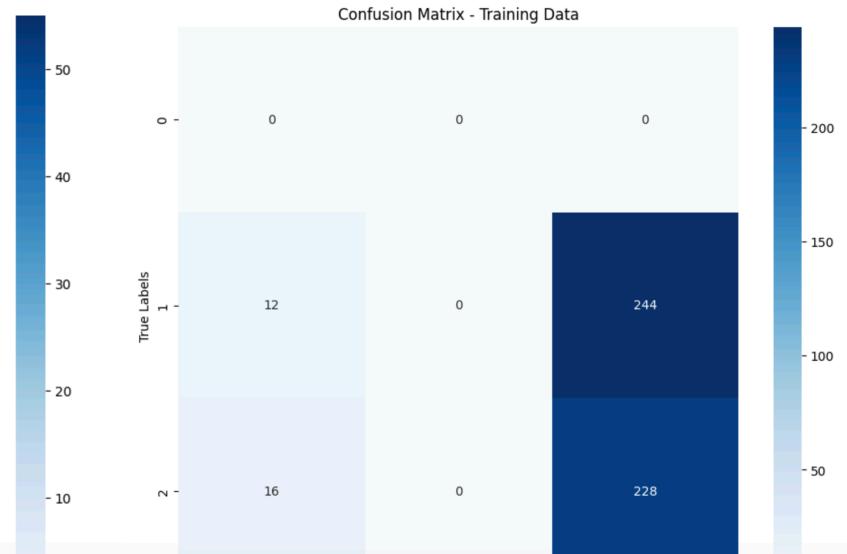
Testing Accuracy: 37.00%

Confusion Matrix - Test Data



Training Accuracy: 45.60%

Confusion Matrix - Training Data



E) SVM - sklearn: Καθώς η παραπάνω υλοποίηση δεν αποφάνθηκε αποδοτική για ολόκληρα τα σύνολα των κλάσεων, χρησιμοποιήθηκε η έτοιμη υλοποίηση από την sklearn. Συγκεκριμένα, έγιναν δοκιμές στα μισά σύνολα και ύστερα στα ολόκληρα καθώς η χρονική πολυπλοκότητα δεν ήταν μεγάλη. Για να αξιολογηθεί η συμπεριφορά του μοντέλου, υπολογίζεται η απόδοση (accuracy_score) στα δεδομένα ελέγχου αλλά και εκπαίδευσης και σχεδιάζεται ο confusion matrix. Τέλος, σχεδιάζεται ένα σχήμα το οποίο απεικονίζει το όριο απόφασης, τα διανύσματα υποστήριξης, το υπερεπίπεδο και τα περιθώρια ενός SVM σε ένα χώρο 2D μετά την εφαρμογή της PCA για μείωση της διαστατικότητας, εάν τα αρχικά δεδομένα έχουν περισσότερα από δύο χαρακτηριστικά. Τα αρχικά σημεία δεδομένων έχουν χρώμα με βάση τις ετικέτες των κλάσεων, το όριο απόφασης απεικονίζεται ως γραμμή περιγράμματος, τα διανύσματα υποστήριξης επισημαίνονται με κόκκινες ακμές, το υπερεπίπεδο αναπαρίσταται με διακεκομμένη γραμμή και τα περιθώρια απεικονίζονται ως διακεκομμένες γραμμές εκατέρωθεν του υπερεπιπέδου.

Στο αρχείο .ipynb περιλαμβάνονται για πληρότητα όλες οι περιπτώσεις.

Παρακάτω ακολουθεί ο πίνακας με τα αποτελέσματα που φέρει το μοντέλο.

Size train data	Size test data	Kernel	C	Training time	Accuracy on train set	Accuracy on test set	Support vectors
5000	1000	Linear	1.0	92.38	93.16%	79.6%	1965
5000	1000	Linear	0.5	64.02	91.92%	80.30%	2022
5000	1000	Linear	2.0	101.2	94.7%	78.9%	1885
10000	2000	Linear	1.0	327.15	89.13%	80.05%	3999
10000	2000	Linear	2.0	473.15	89.93%	79.65%	3940
10000	2000	Polynomial (d=1)	1.0	154.65	83.51%	83.0%	4626
10000	2000	Polynomial (d=1)	2.0	149.95	83.92%	82.35%	4503
10000	2000	Polynomial (d=1)	0.5	147.8	83.05%	83.1%	4800

10000	2000	Polynomial (d=2)	1.0	126.04	93.52%	90.15%	3816
10000	2000	Polynomial (d=2)	2.0	119.71	95.8%	90.3%	3642
10000	2000	Polynomial (d=3)	1.0	117.75	99.41%	91.45%	3492
10000	2000	Polynomial (d=4)	1.0	114.97	99.41%	91.45%	3492
10000	2000	Gaussian	1.0	114.48	94.31%	90.4%	4528
10000	2000	Gaussian	2.0	117.08	96.95%	91.6%	4181

1η εκτέλεση :

```

Training Time: 92.38 seconds
SVM Parameters:
Kernel Type: linear
Regularization Parameter (C): 1.0
Training Accuracy: 93.16%
Testing Accuracy: 79.60%
Examples of Correct Classifications on Training Set:
Input: [[0.6666667  0.7058824  0.7764706 ]
 [0.65882355 0.69803923 0.76862746]
 [0.69411767 0.7254902 0.79607844]
 ...
 [0.63529414 0.7019608 0.84313726]
 [0.61960787 0.69803923 0.8392157 ]
 [0.6156863 0.69411767 0.83137256]]
[[0.65882355 0.70980394 0.7764706 ]
 [0.6745098 0.7254902 0.7882353 ]
 [0.67058825 0.7176471 0.78431374]
 ...
 [0.62352943 0.69411767 0.83137256]
 [0.6117647 0.6901961 0.827451 ]
 [0.6039216 0.68235296 0.819607851]
 ...
 [[0.6039216 0.6666667 0.7294118 ]
 [0.58431375 0.64705884 0.70980394]
 [0.5058824 0.5647059 0.63529414]
 ...
 [0.6313726 0.69803923 0.8392157 ]
 [0.6156863 0.69411767 0.83137256]
 [0.6039216 0.68235296 0.819607851]
 ...

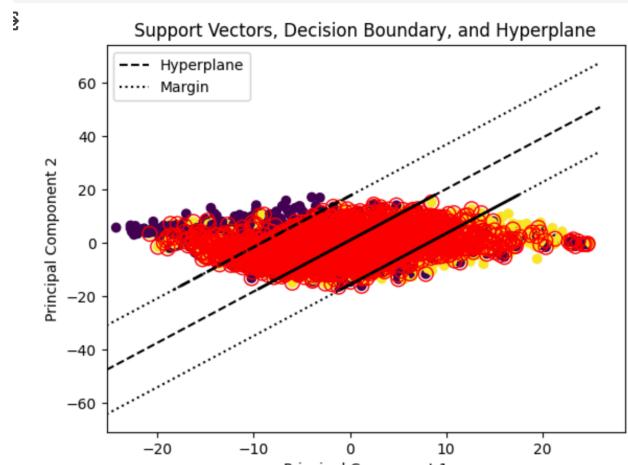
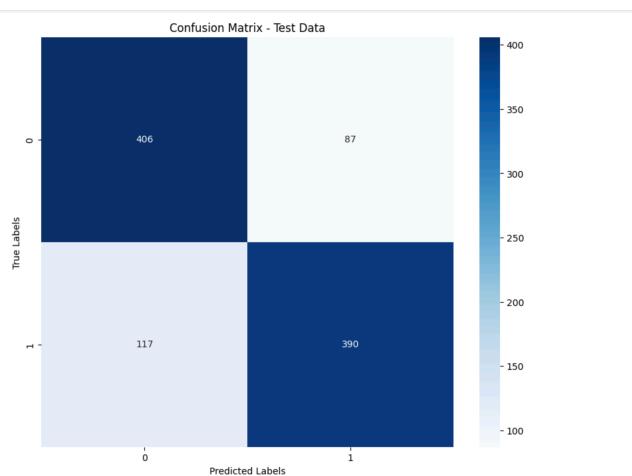
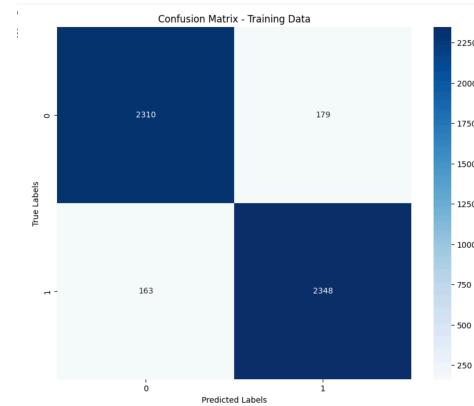
```

```

support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])

```

Number of Support Vectors: 1965

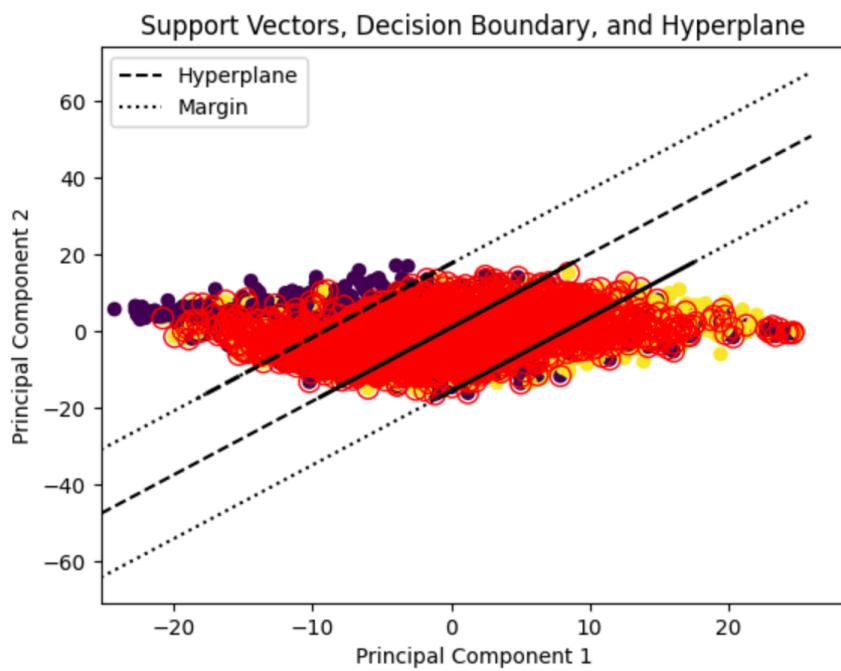
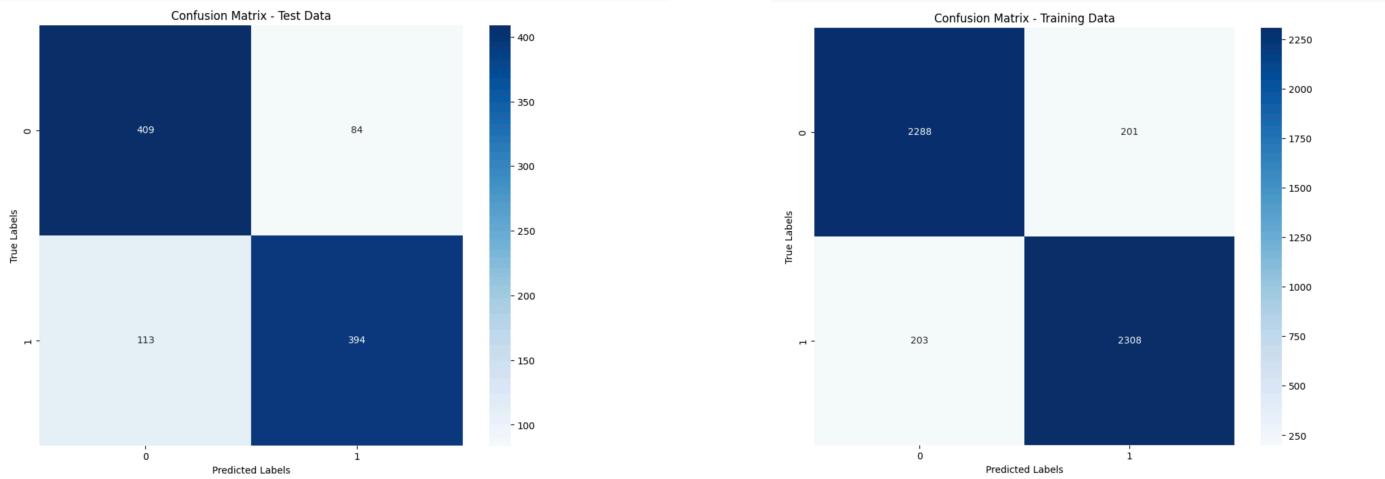


2η εκτέλεση :

Training Time: 64.02 seconds
SVM Parameters:
Kernel Type: linear
Regularization Parameter (C): 0.5
Training Accuracy: 91.92%
Testing Accuracy: 80.30%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 2022

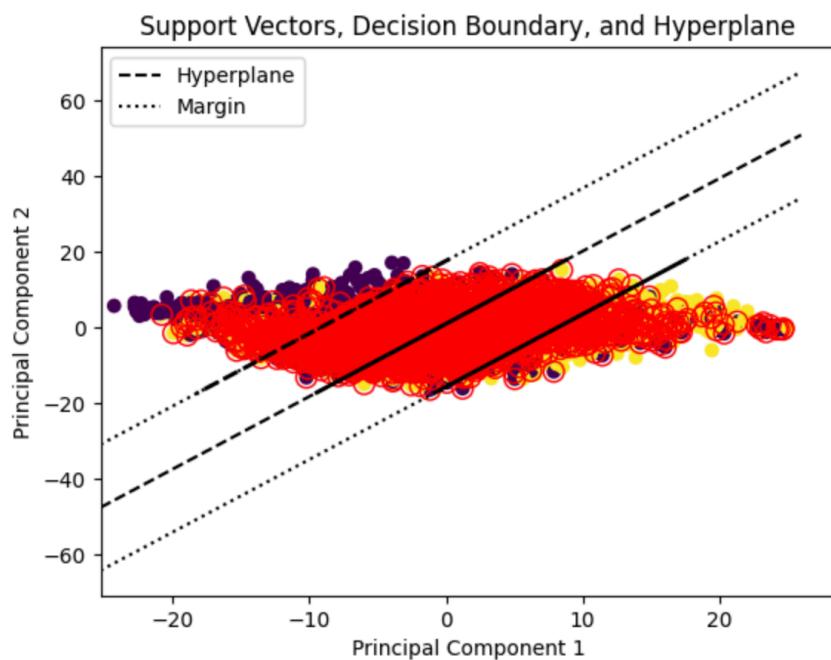
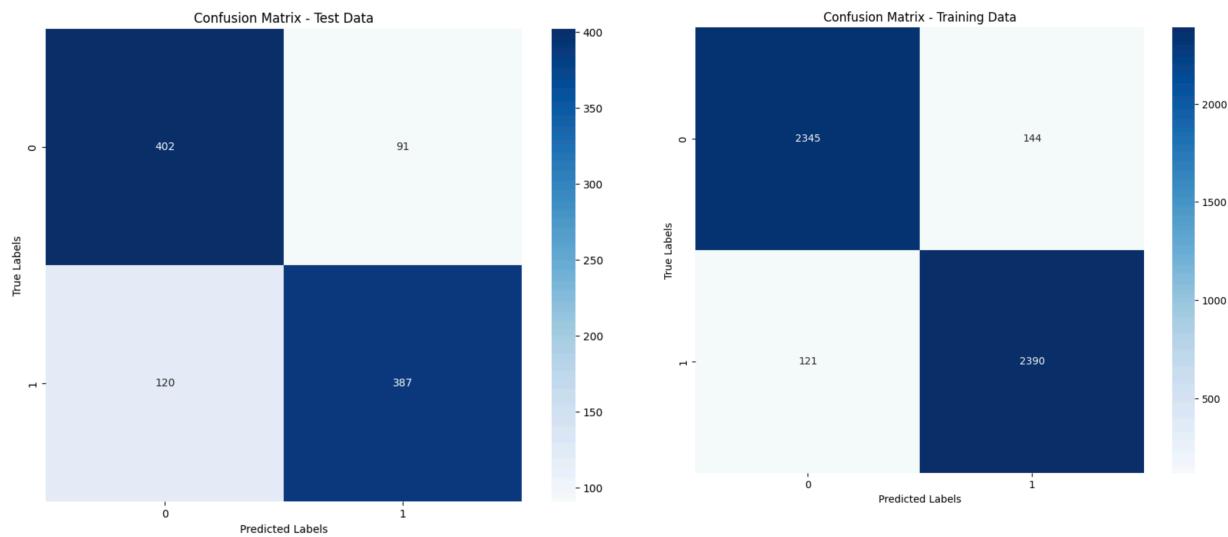


3η εκτέλεση :

Training Time: 101.20 seconds
SVM Parameters:
Kernel Type: linear
Regularization Parameter (C): 2.0
Training Accuracy: 94.70%
Testing Accuracy: 78.90%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

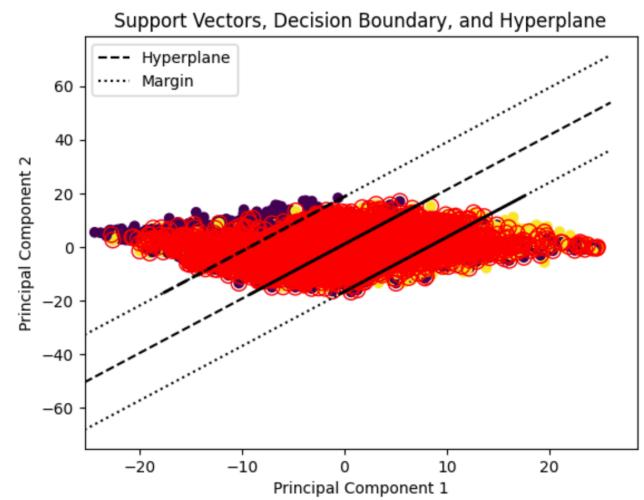
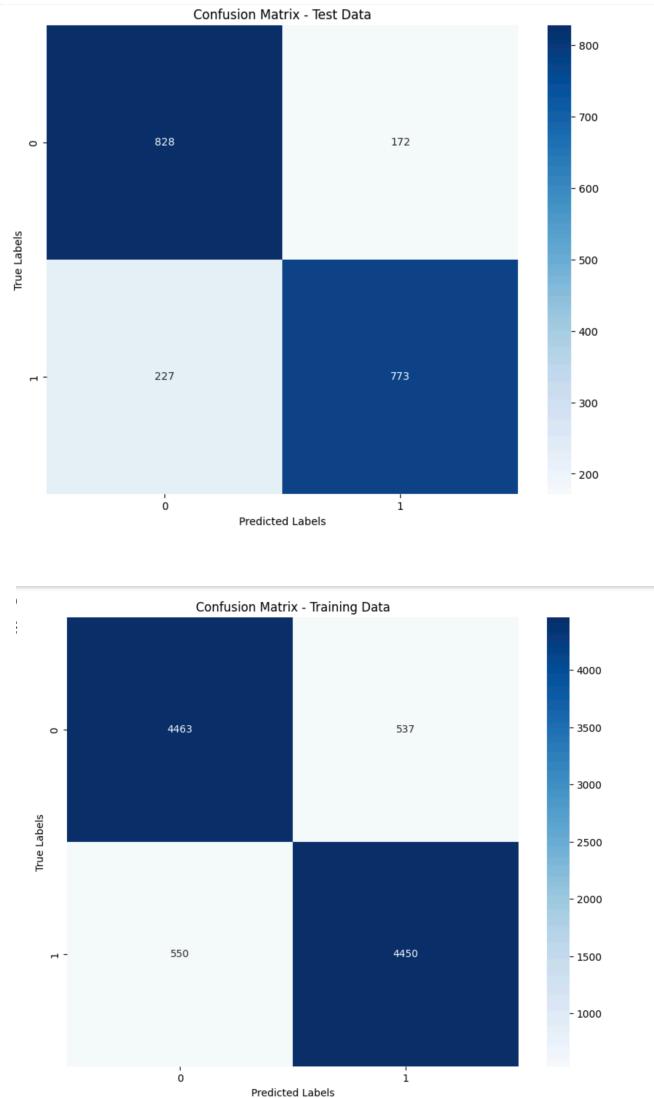
Number of Support Vectors: 1885



4η εκτέλεση :

Training Time: 327.15 seconds
SVM Parameters:
Kernel Type: linear
Regularization Parameter (C): 1.0
Training Accuracy: 89.13%
Testing Accuracy: 80.05%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
Number of Support Vectors: 3999
```

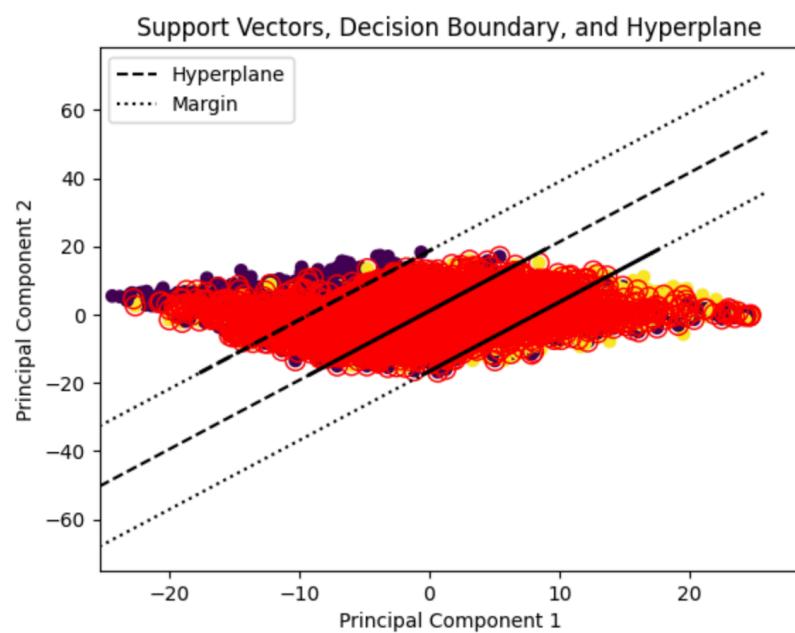
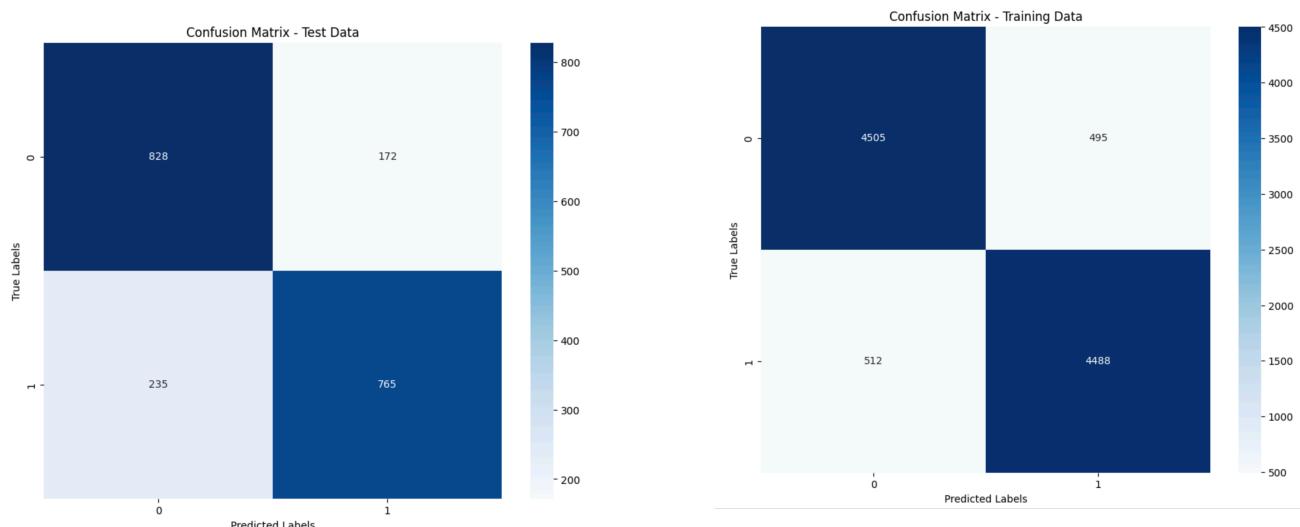


5η εκτέλεση :

Training Time: 473.15 seconds
SVM Parameters:
Kernel Type: linear
Regularization Parameter (C): 2.0
Training Accuracy: 89.93%
Testing Accuracy: 79.65%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 3940

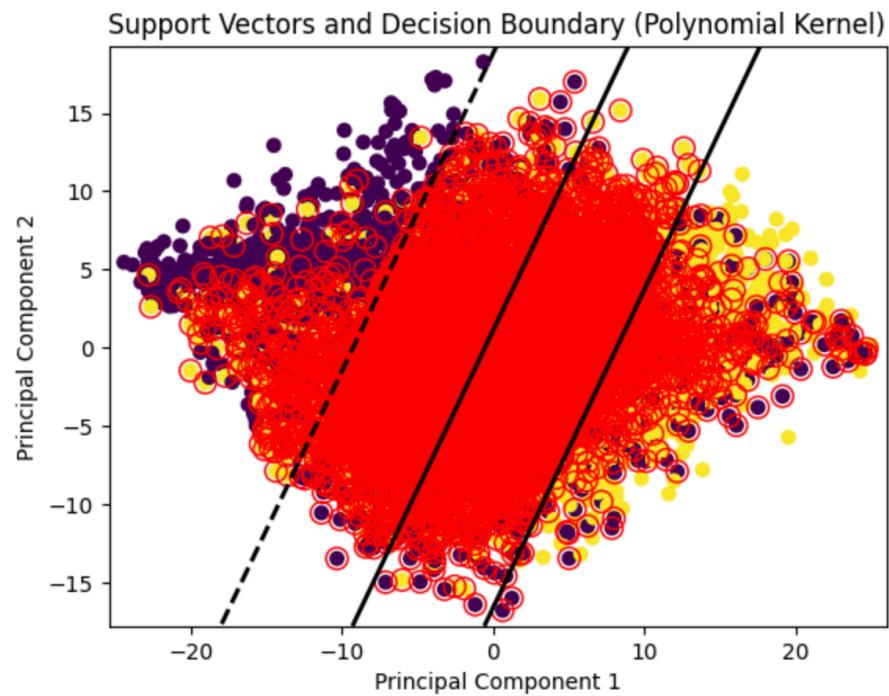
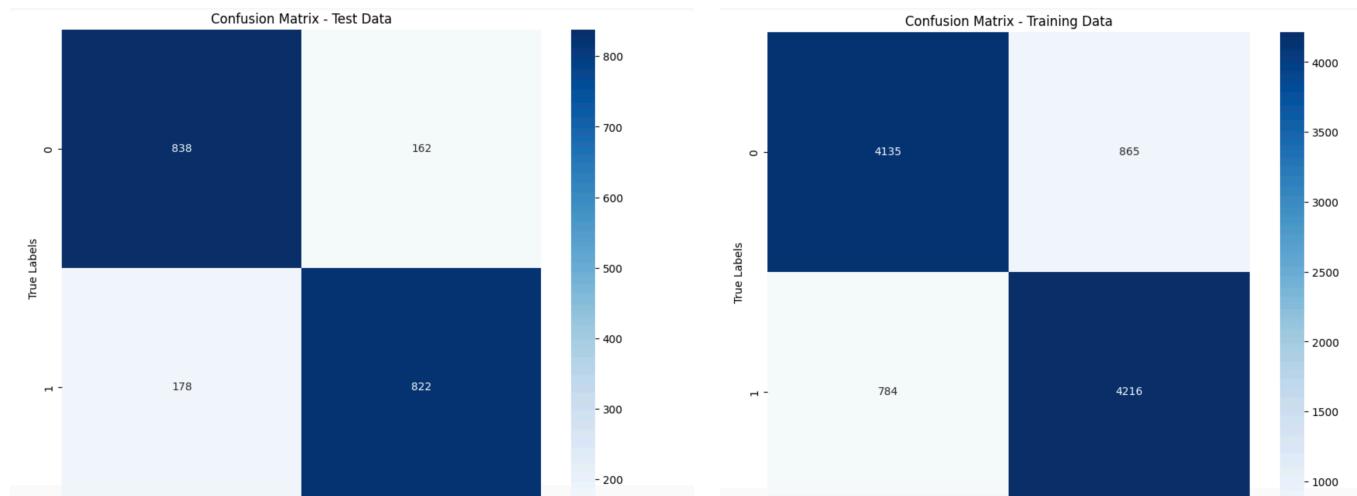


6η εκτέλεση :

Training Time: 154.65 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 1.0
Training Accuracy: 83.51%
Testing Accuracy: 83.00%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 4626

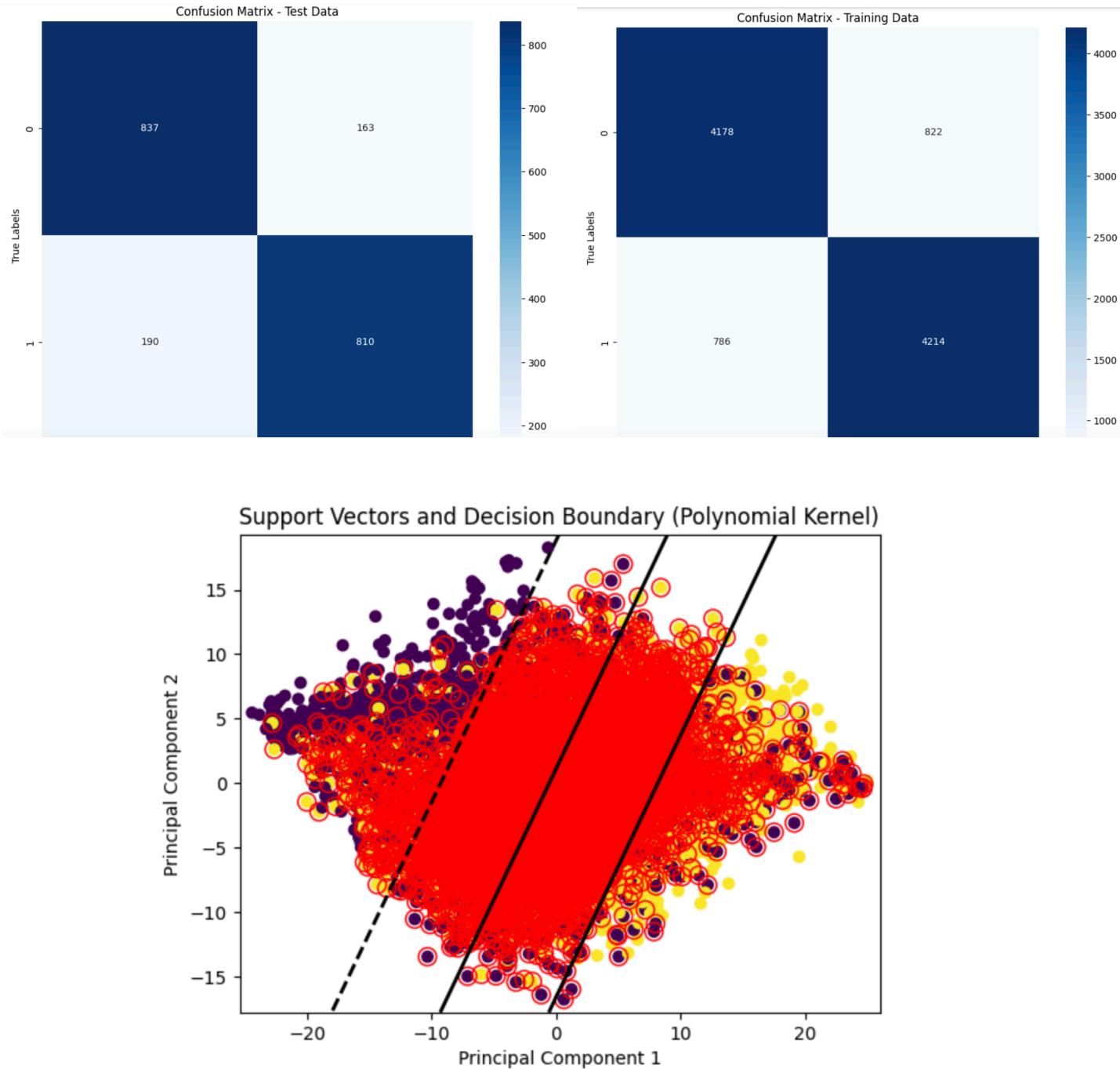


7η εκτέλεση :

Training Time: 149.95 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 2.0
Training Accuracy: 83.92%
Testing Accuracy: 82.35%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 4503

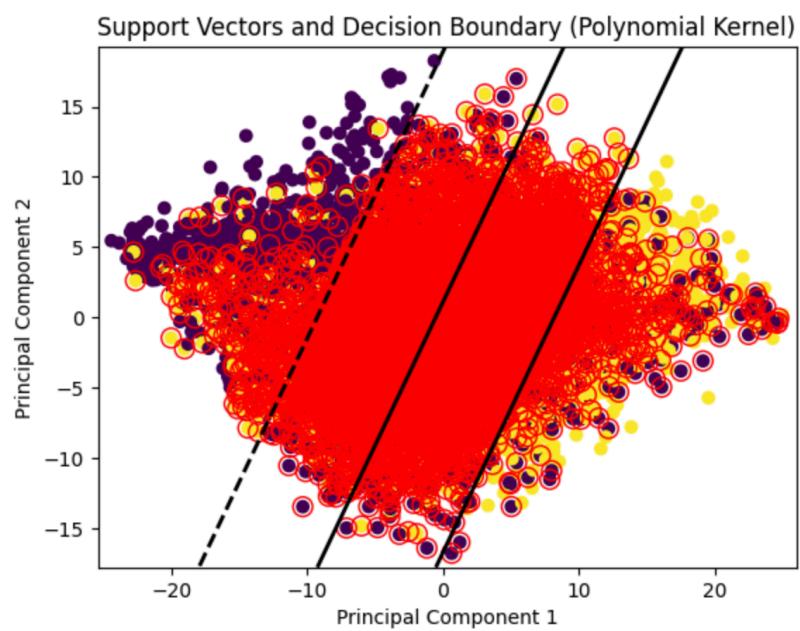
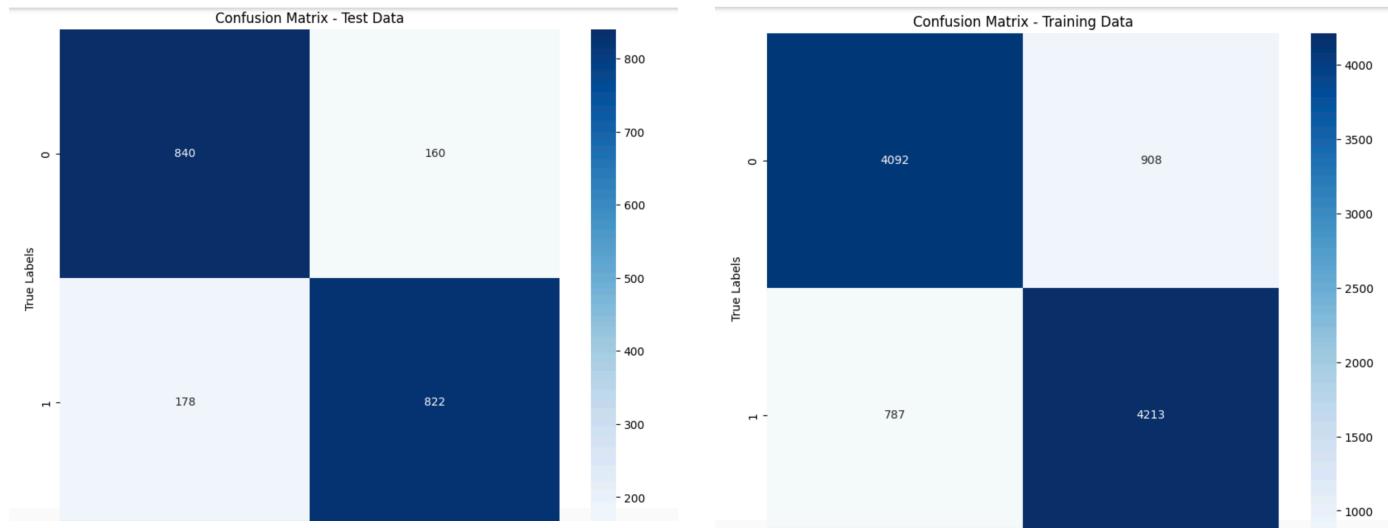


8η εκτέλεση :

Training Time: 147.80 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 0.5
Training Accuracy: 83.05%
Testing Accuracy: 83.10%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 4800

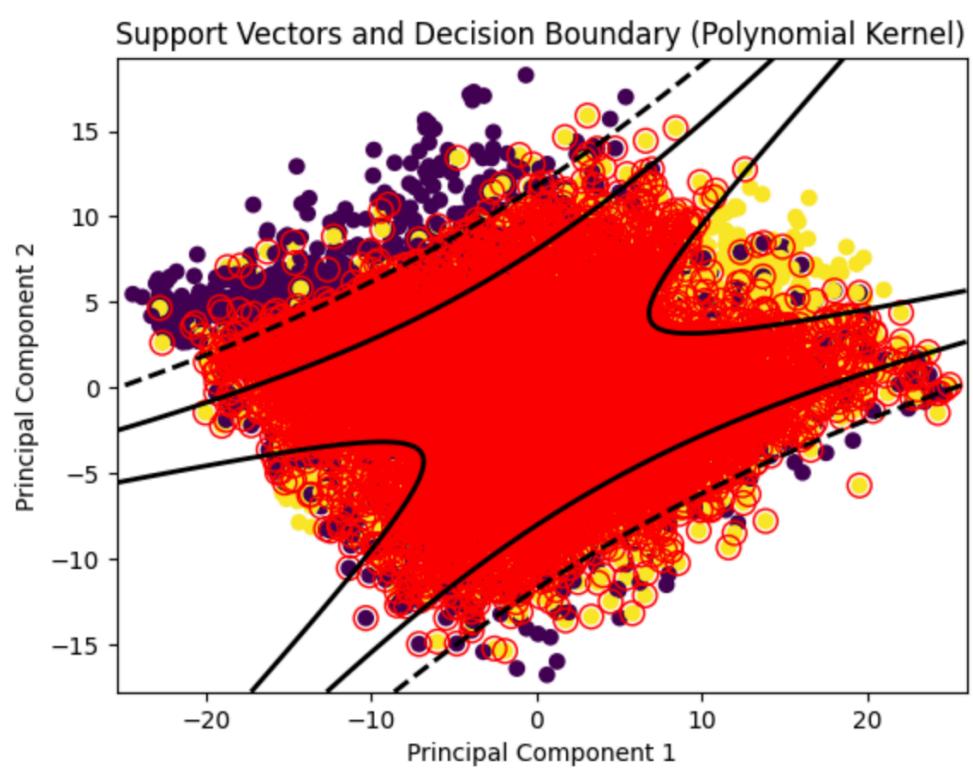
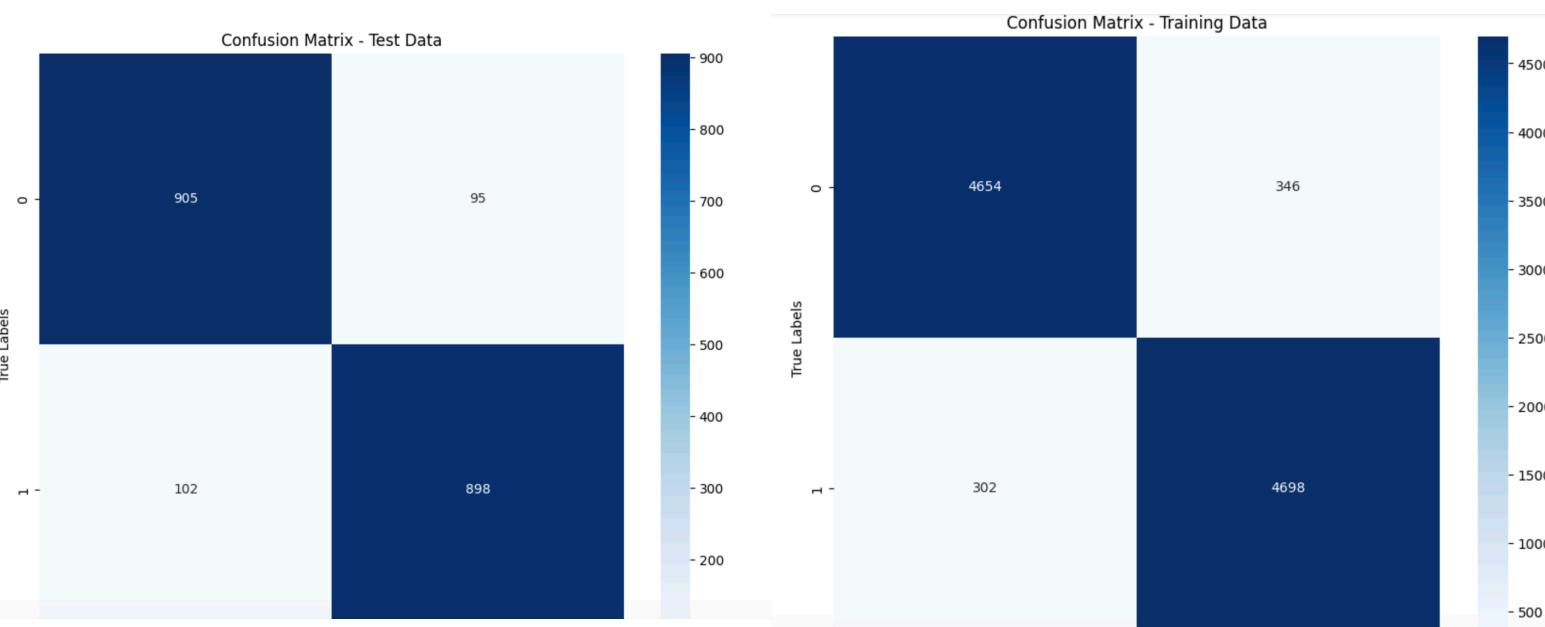


9η εκτέλεση :

Training Time: 126.04 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 1.0
Training Accuracy: 93.52%
Testing Accuracy: 90.15%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 3816

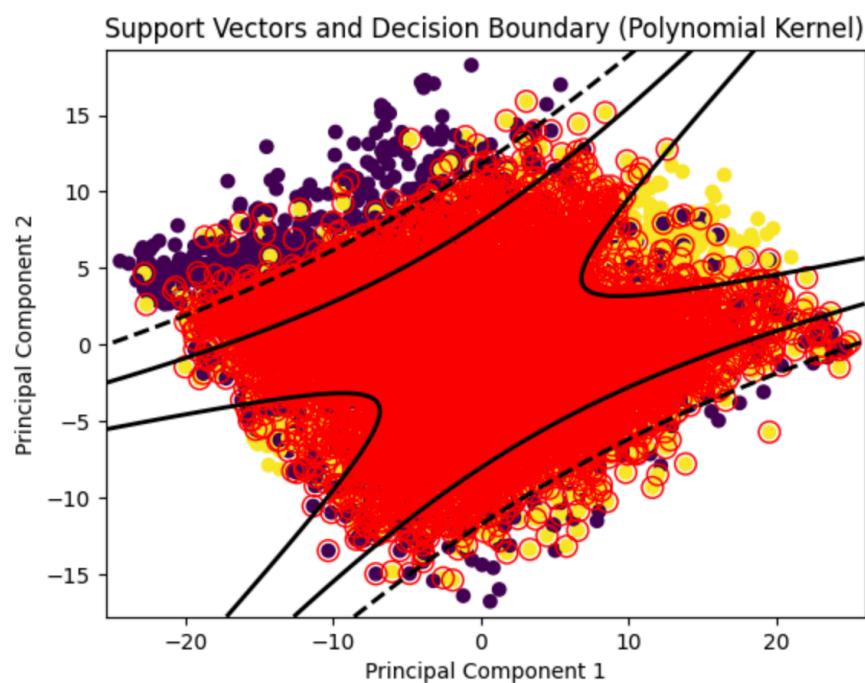
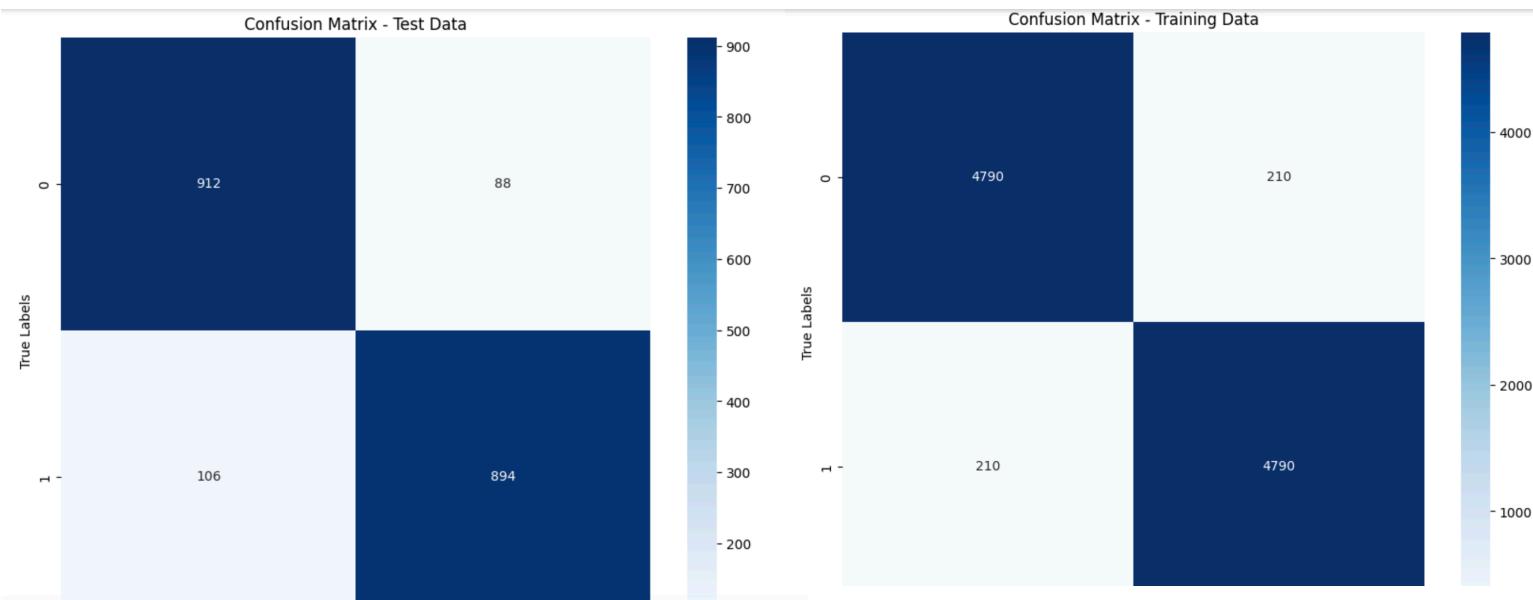


10η εκτέλεση :

Training Time: 119.71 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 2.0
Training Accuracy: 95.80%
Testing Accuracy: 90.30%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 3642



11η εκτέλεση :

Training Time: 117.75 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 1.0
Training Accuracy: 99.41%
Testing Accuracy: 91.45%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 3492

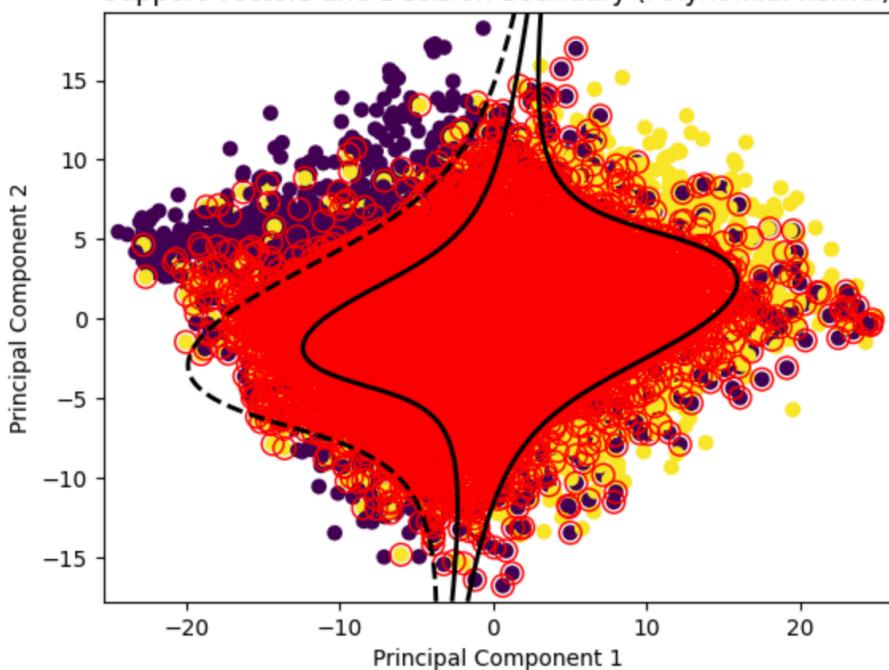
Confusion Matrix - Test Data



Confusion Matrix - Training Data



Support Vectors and Decision Boundary (Polynomial Kernel)

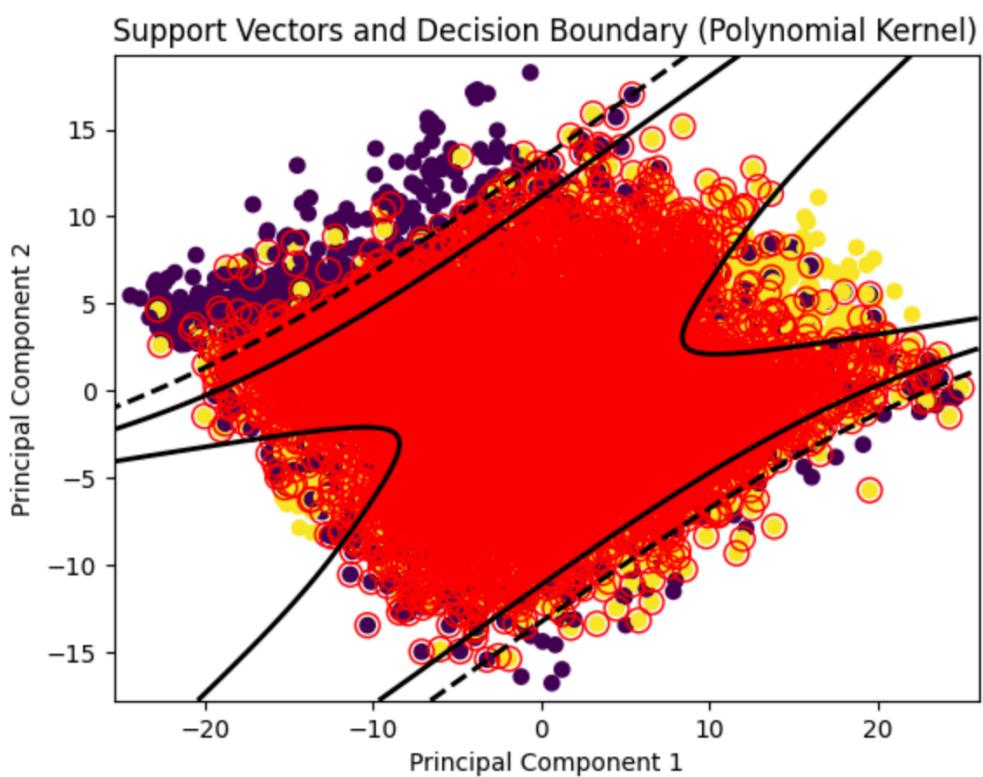
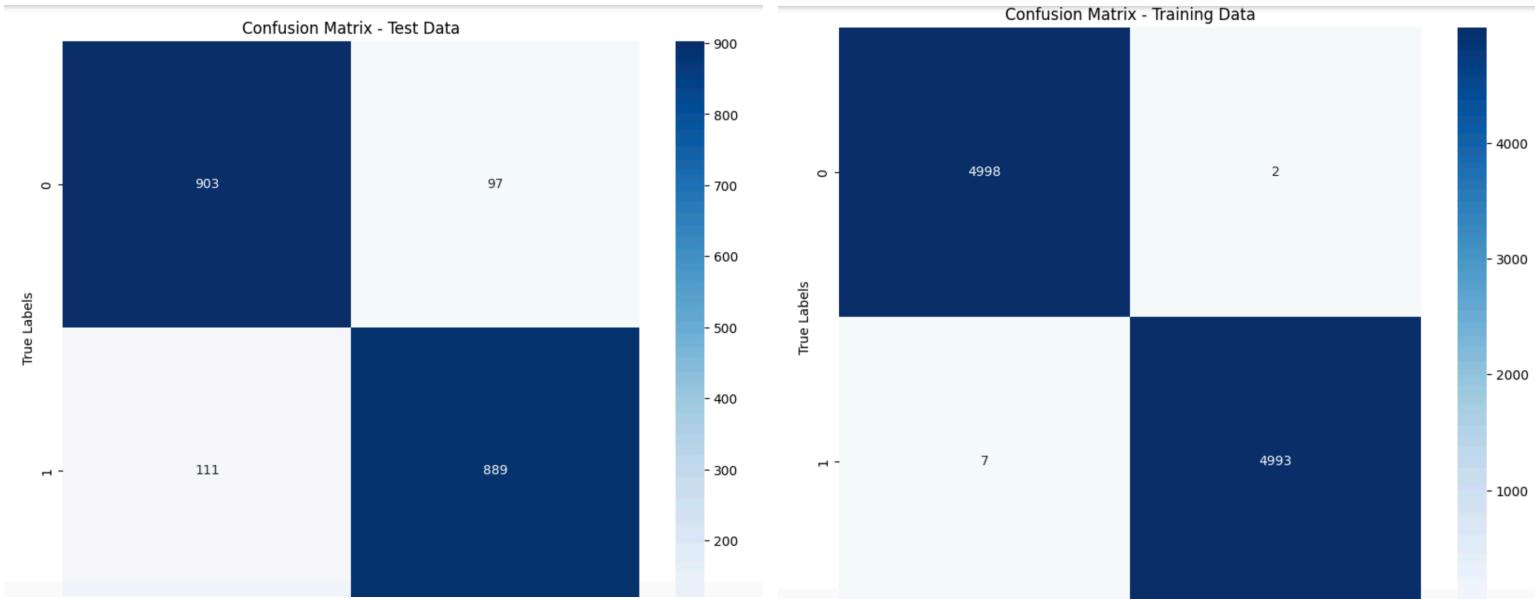


12η εκτέλεση :

Training Time: 102.99 seconds
SVM Parameters:
Kernel Type: poly
Regularization Parameter (C): 1.0
Training Accuracy: 99.91%
Testing Accuracy: 89.60%

```
support_vectors = svm_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 3173

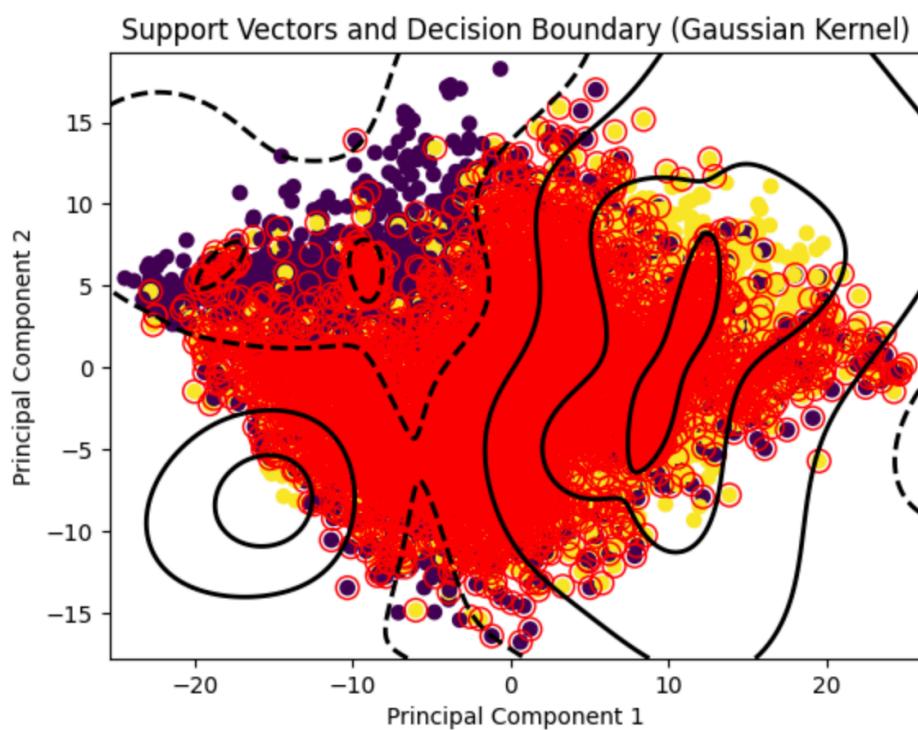
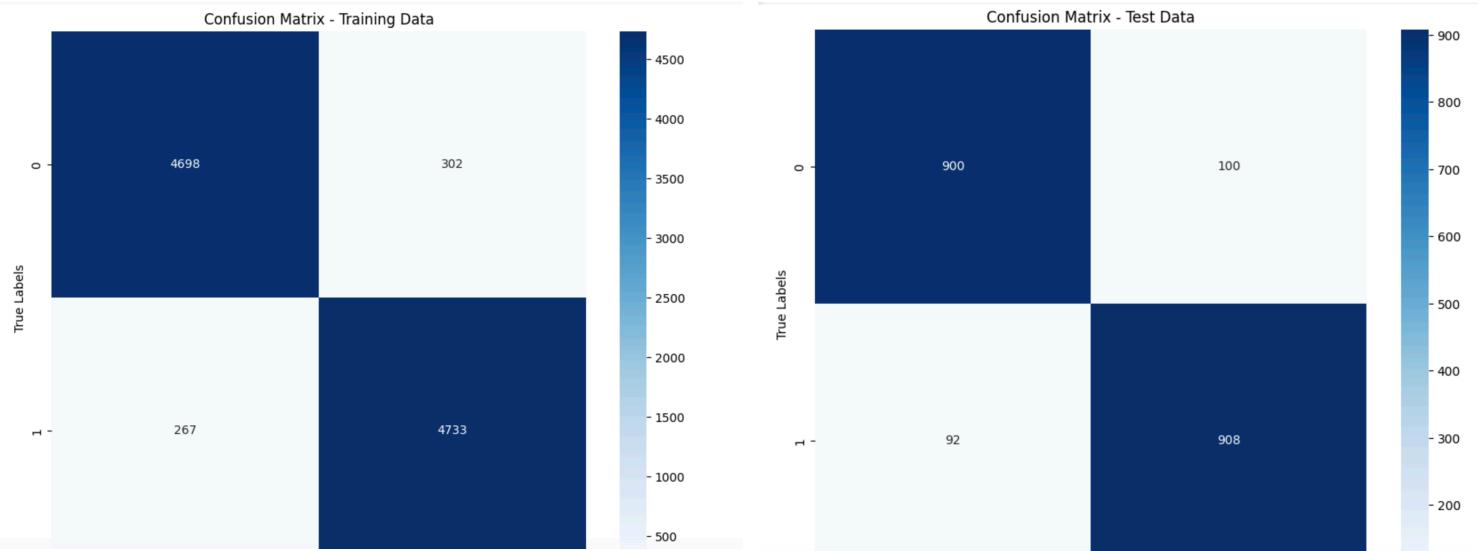


13η εκτέλεση :

Training Time: 114.48 seconds
SVM Parameters:
Kernel Type: rbf
Regularization Parameter (C): 1.0
Training Accuracy: 94.31%
Testing Accuracy: 90.40%

```
support_vectors = rbf_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 4258

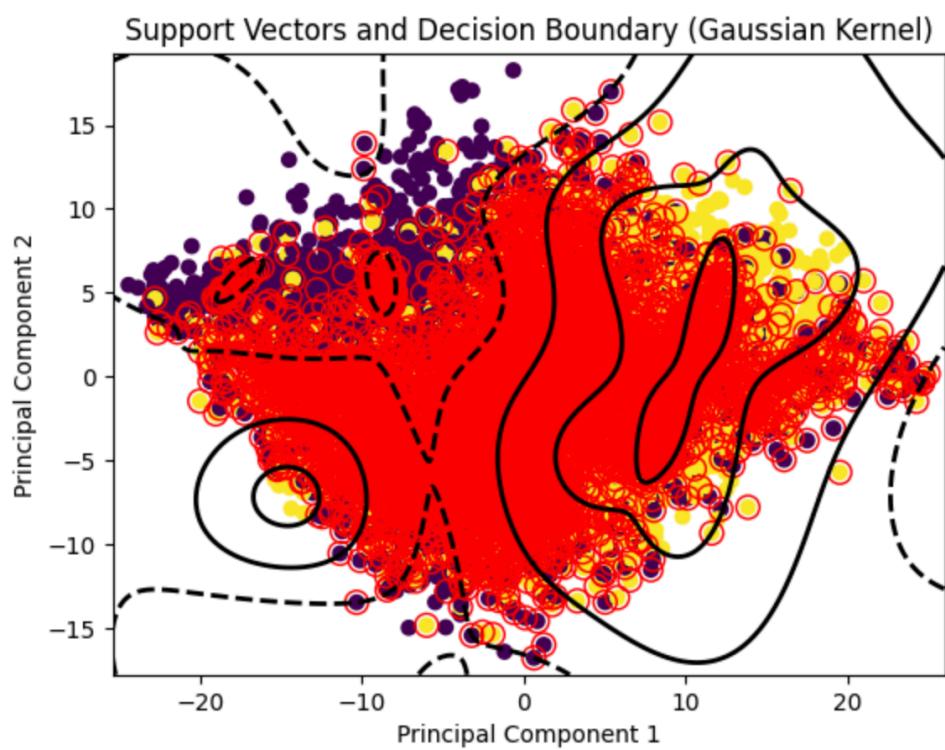
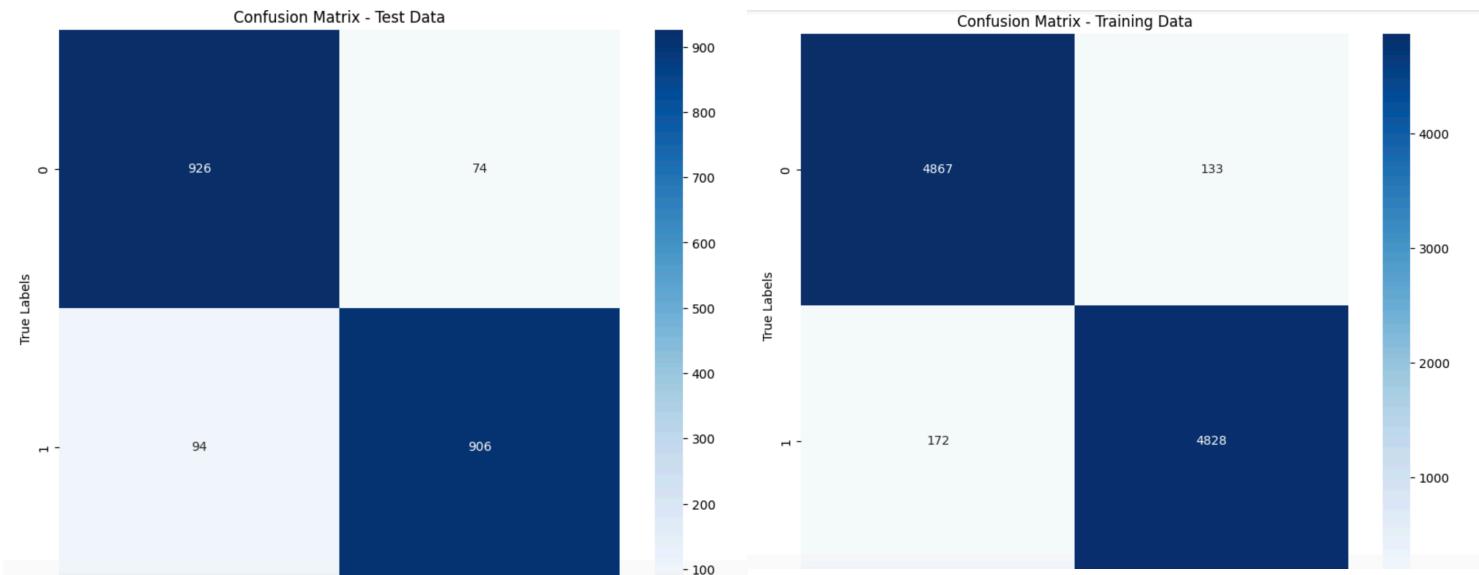


14η εκτέλεση :

Training Time: 117.08 seconds
SVM Parameters:
Kernel Type: rbf
Regularization Parameter (C): 2.0
Training Accuracy: 96.95%
Testing Accuracy: 91.60%

```
support_vectors = rbf_model.support_vectors_
print("Number of Support Vectors:", support_vectors.shape[0])
```

Number of Support Vectors: 4181



ΣΤ) Συμπεράσματα - Παρατηρήσεις: Στην πρώτη υλοποίηση, καλύτερη απόδοση σημειώνεται με τον γραμμικό πυρήνα όταν έχει $\epsilon = 0.001$, $\text{max_iter} = 100$ και $C = 2$. Αυξάνοντας την τιμή κατωφλίου ή μειώνοντας την τιμή στον συντελεστή κανονικοποίησης, η απόδοση πέφτει. Δυστυχώς, τα δεδομένα είναι πολύ λίγα για να στηριχτούμε σε γενικά συμπεράσματα λαμβάνοντας υπόψιν μόνο αυτήν την υλοποίηση.

Στην δεύτερη υλοποίηση, συγκρίνοντας τις εκτελέσεις που αφορούν ολόκληρα τα σύνολα των 2 κλάσεων, προέκυψαν τα εξής συμπεράσματα:

1. Συγκρίνοντας τις δοκιμές με γραμμικό πυρήνα ως προς την απόδοση στα δεδομένα εκπαίδευσης, μεγαλύτερο ποσοστό (με μικρή διαφορά) επιτυγχάνεται με συντελεστή κανονικοποίησης ίσο με 2.
2. Συγκρίνοντας τις δοκιμές είτε με γραμμικό είτε με πολυωνυμικό είτε με `gaussian` πυρήνα, τα αποτελέσματα είναι καλύτερα όταν χρησιμοποιείται ο δεύτερος και ο τρίτος από αυτούς. Ωστόσο, στις συγκεκριμένες εκτελέσεις τα ποσοστά είναι εξαιρετικά μεγάλα, οδηγώντας στην υπόθεση πως έγινε overfitting, ιδίως στους πολυωνυμικούς πυρήνες με βαθμό 3 και 4 και `guassian`. Όσον αφορά τον συντελεστή κανονικοποίησης, μειώνοντας την τιμή του, τα ποσοστά πέφτουν ενώ αυξάνοντας την, μεγαλώνουν.

Z) Πηγές - Chatgpt: Στην πρώτη υλοποίηση όπου έγινε προσπάθεια να δημιουργηθεί η `svm` από την αρχή, χρησιμοποιήθηκε η λογική του SMO αλγορίθμου, λαμβάνοντας πληροφορίες περί αυτού από την εξής πηγή: <http://www.big-data.tips/sequential-minimal-optimization>

Σχετικά με το `chatgpt`, χρησιμοποιήθηκε σχετικά με την απεικόνιση των αποτελεσμάτων. Συγκεκριμένα, σε σχετική ερώτηση πρότεινε την λύση του `confusion matrix`, όπου και χρησιμοποιήθηκε. Ύστερα, έμμεσα χρησιμοποιήθηκε στην πρώτη υλοποίηση όσον αφορά την απόδοση: παρατηρώντας τα αποτελέσματα στο τέλος των επαναλήψεων και στα `confusion matrices` σημειώνεται μικρή αριθμητική διαφορά. Ρωτώντας την συγκεκριμένη μηχανή για το κομμάτι αυτό, ως πιθανούς λόγους, πρότεινε: Τυχαία δεδομένα: Η συνάρτηση `get_rnd_int` χρησιμοποιείται για την επιλογή τυχαίων δεικτών κατά τη διάρκεια της εκπαίδευσης. Εάν

η τυχαιοποίηση δεν είναι ελεγχόμενη ή δεν είναι seeded, διαφορετικές εκτελέσεις της διαδικασίας εκπαίδευσης μπορεί να οδηγήσουν σε διαφορετικά σύνολα τυχαίων δεικτών και, κατά συνέπεια, σε διαφορετικές ενημερώσεις του μοντέλου. Αρχικοποίηση: Οι αρχικές τιμές των παραμέτρων του μοντέλου (π.χ. alpha) μπορεί να διαφέρουν μεταξύ των εκτελέσεων, οδηγώντας σε διαφορετικές πορείες σύγκλισης κατά την εκπαίδευση. Επιλογή διανύσματος υποστήριξης: Τα διανύσματα υποστήριξης επιλέγονται κατά τη διάρκεια της εκπαίδευσης και το σύνολο των διανυσμάτων υποστήριξης ενδέχεται να διαφέρει μεταξύ των εκτελέσεων, επηρεάζοντας το όριο απόφασης και, κατά συνέπεια, την ακρίβεια. Ακρίβεια κινητής υποδιαστολής: Λόγω της αριθμητικής κινητής υποδιαστολής, ενδέχεται να συσσωρευτούν μικρές αριθμητικές διαφορές, ιδίως όταν πρόκειται για επαναληπτικούς αλγορίθμους βελτιστοποίησης.

Στα τελικά αποτελέσματα, λοιπόν, προστίθενται τα ποσοστά που λαμβάνονται από την μετρική accuracy_score.

Η) Σύγκριση με τους KNN, NC: Για παραπάνω διερεύνηση και αξιολόγηση του μοντέλου, χρησιμοποιήθηκαν και οι classifiers κ-κοντινότερων γειτόνων και κοντινότερου κέντρου. Όπως φαίνεται παρακάτω, συγκρίνοντας με την δεύτερη υλοποίηση του svm (καθώς στους KNN, NC χρησιμοποιούνται ολόκληρα τα σύνολα των 2 κλάσεων), το svm κατορθώνει καλύτερα αποτελέσματα.

```

15.  ✓ from sklearn.neighbors import NearestCentroid
     from sklearn.metrics import accuracy_score

     nc_model = NearestCentroid()
     nc_model.fit(x_train_flattened, np.ravel(y_train_subset))

     train_predictions_nc = nc_model.predict(x_train_flattened)
     train_accuracy_nc = accuracy_score(y_train_subset, train_predictions_nc)
     print("Training Accuracy (Nearest Centroid): {:.2%}".format(train_accuracy_nc))

     test_predictions_nc = nc_model.predict(x_test_flattened)
     test_accuracy_nc = accuracy_score(y_test_subset, test_predictions_nc)
     print("Testing Accuracy (Nearest Centroid): {:.2%}".format(test_accuracy_nc))

      ↗ Training Accuracy (Nearest Centroid): 70.76%
      ↗ Testing Accuracy (Nearest Centroid): 72.30%


36.  ✓ from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score

     knn_model = KNeighborsClassifier(n_neighbors=2)
     knn_model.fit(x_train_flattened, np.ravel(y_train_subset))

     train_predictions_knn = knn_model.predict(x_train_flattened)
     train_accuracy_knn = accuracy_score(y_train_subset, train_predictions_knn)
     print("Training Accuracy (KNN): {:.2%}".format(train_accuracy_knn))

     test_predictions_knn = knn_model.predict(x_test_flattened)
     test_accuracy_knn = accuracy_score(y_test_subset, test_predictions_knn)
     print("Testing Accuracy (KNN): {:.2%}".format(test_accuracy_knn))

      ↗ Training Accuracy (KNN): 74.14%
      ↗ Testing Accuracy (KNN): 63.40%

```