

Νευρωνικά Δίκτυα - Εργασία 2

Ρούσος Σταμάτης

22 Ιανουαρίου 2026

Εισαγωγή

Σε αυτή την εργασία ασχολουμαστε με τα SVM (Support Vector Machines) ως κατηγοριοποιητες. Εσπασα την εργασία σε 3 μέρη.

1. Στο 1ο Μέρος, εψαξα να βρω το καλύτερο SVM, κρίνοντας με βάση την αποδοση εκπαίδευσης και γενικευσης (training/validation accuracy).
2. Στο 2ο Μέρος, παίρνω το καλύτερο SVM που βρήκα, και το συγκρίνω με ένα MLP Neural Network.
3. Στο 3ο Μέρος, πήρα το καλύτερο CNN που υλοποίησα στην 1η εργασία, και αντί για το τελευταίο layer κατηγοριοποίησης που είχε, χρησιμοποίησα το καλύτερο SVM μου ως κατηγοριοποιητή.

Προκειμένου να δοκιμάσω πολλά μοντέλα SVM με διαφορετικές τιμές παραμετρών, χρησιμοποίησα την `cuml` για το SVM, η οποία χρησιμοποιεί GPU και έτσι οι υπολογισμοί γίνονταν πολύ πιο γρήγορα (λόγω παραλληλίας). Απλά για να το κάνω αυτό, χρειάστηκε να μετατρέψω τα δεδομένα σε συγκεκριμένο είδος array για την GPU, το οποίο έκανα με την `cupy`. Αυτά έγιναν με τη βοήθεια του ChatGPT και συναδέλφου που ρώτησα.

Προεπεξεργασία Δεδομένων

Το dataset που χρησιμοποίησα είναι το CIFAR-10 το οποίο αποτελείται συνολικά από 60.000 έγχρωμες εικόνες διαστάσεων $32 \times$

32 με 3 κανάλια RGB. Αρα καθε εικονα αποτελείται $32 \times 32 \times 3 = 3072$ τιμες-διαστασεις οι οποιες αναπαριστουν $[R \ G \ B]$ τιμες καθενος απο τα 32×32 πιξελ μιας εικονας. Οι 10.000 εικονες ει-ναι ηδη διαχωρισμενες, αποτελουν το test set και χρησιμοποιου-νται αποκλειστικα για την αξιολογηση του τελικου μοντελου.

Για την σωστη αξιολογηση καθε μοντελου χρησιμοποιησα ενα validation set των 10.000 εικονων με την συναρτηση train-test-split της sklearn. Ετσι, εμειναν 40.000 εικονες για την εκπαιδευση. Οι τιμες των pixels κανονικοποιηθηκαν διαιρωντας με 255 ωστε να ει-ναι ολες στην ιδια κλιμακα (διαστημα $[0,1]$).

Στη συνεχεια, εκανα Scaling, με την StandardScaler και τις εντο-λες fit,transform. Η fit, υπολογιζει το μεσο ορο και την τυπικη απο-κλιση των 40.000 εικονων εκπαιδευσης, και η transform εφαρμοζει το μετασχηματισμο:

$$z = \frac{x - \mu}{\sigma}$$

Προσοχη! Το fit γινεται μονο στα training data, και τα validation-test data κανουν μονο transform με τον μεσο ορο και την τυπικη αποκλιση του training dataset. Αυτο γιατι, οπως εγραψα και στο notebook, αν εκανα fit και στο validation/test set, τοτε θα ηταν σαν να τα θεωρουσα δεδομενα εκπαιδευσης.

Τελος, στο SVM χρησιμοποιησα PCA για τη μειωση των διαστα-σεων.

Θεωρία Support Vector Machines

Εστω οτι ειμαστε σε δυο διαστασεις και εχουμε ενα συνολο δειγματων που ανηκουν σε δυο διαφορετικες κλασεις. Θελουμε να βρουμε ενα τροπο ωστε να κατηγοριοποιουμε σωστα οποιοδηποτε νεο (αγνωστο) δειγμα.

Τα γραμμικα Support Vector Machines (SVM) προσπαθουν να λυσουν αυτο το προβλημα γραμμικα, δηλαδη αναζητωντας μια δια-

χωριστική γραμμή η οποία να διαχωρίζει σωστά τα δείγματα των δυο κλάσεων. Ομως, γενικά υπάρχουν απειρες τετοιες ευθειες (αν το dataset είναι γραμμικά διαχωρισίμο).

Η βελτιστη διαχωριστική ευθεία, δηλαδή η ευθεία που επιτρέπει την καλύτερη-ευκολότερη ταξινόμηση χωρίς λαθη (πχ σημεία κοντα στο οριο ή που έχουν περασει λιγο το οριο αλλά είναι πιο κοντα στην δική τους κλάση), το SVM επιλέγει εκείνη τη γραμμή που μεγιστοποιεί την ελάχιστη αποσταση όλων των σημειων από αυτήν. Ισοδυναμα, αρκει να μεγιστοποιησουμε τις αποστασεις των κοντινότερων σημειων των δυο κλάσεων απο τη διαχωριστική γραμμή. Τα σημεία αυτά ονομαζονται *support vectors* και είναι τα μονα δείγματα που επηρεάζουν τη θέση της διαχωριστικής γραμμής.

Η αποσταση των support vectors απο τη διαχωριστική γραμμή ονομαζεται *margin* και αυτο προσπαθει να μεγιστοποιησει το SVM.

Επειδη το hard margin απαιτει τελειο διαχωρισμο και δεν επιτρεπει λαθη, κανει παρα πολυ δυσκολη την ευρεση της διαχωριστικής γραμμής. Γι' αυτο το λογο, συνηθως χρησιμοποιουμε soft margin σε δεδομενα που είναι δυσκολα διαχωρισίμα (πχ πολλές διαστασεις ή πολλά δείγματα κλπ). Το soft Margin επιτρεπει λαθη, προσθετοντας ενα παραγοντα χαλαρωσης. Αυτο βοηθαει στην ευρεση καλύτερης λύσης του προβληματος. Σε datasets που δεν είναι γραμμικά διαχωρισίμα (οπως στο δικο μας), μεταφερουμε τα δεδομενα σε υψηλότερη διασταση, με σκοπο να γινουν γραμμικά διαχωρισίμα. Αυτο το κανουμε με τη χρηση καποιας συναρτησης χαρτογραφησης $\phi(x)$ που παιρνει καθε δείγμα και το μετασχηματίζει σε ψηλότερη διασταση. Για παραδειγμα για ενα σημείο $x = (x_1, x_2)$ δυο διαστασεων, η συναρτηση:

$$\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

το μεταφερει στις 3 διαστασεις. Επειδη, ομως, σε μεγαλυτερες δια-

στασεις αυτο ειναι υπολογιστικα πολυ ακριβο (παμε σε υπερβολικα μεγαλες διαστασεις και τα εσωτερικα γινομενα που χρειαζεται να υπολογισει το SVM ειναι τεραστια), με τη χρηση πυρηνων (Kernels) χρησησιμοποιουμε συναρτησεις που δινουν το ιδιο αποτελεσμα εσωτερικων γινομενων, χωρις να παμε στην πραγματικοτητα στην ψηλοτερη διασταση, γλιτωνοντας ετσι αυτον τον τεραστιο υπολογιστικο ογκο.

Μαθηματικά

Το διαχωριστικο υπερεπιπεδο στον αρχικο χωρο δινεται απο τη σχεση:

$$w^T x + b = 0.$$

Η συναρτηση αποφασης ειναι η:

$$f(x) = w^T x + b,$$

και η προβλεψη:

$$\hat{y} = \text{sign}(f(x)).$$

Το αρχικο optimization προβλημα που καλειται να λυσει το SVM ειναι:

$$\min_{w,b,\xi} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

$$\begin{aligned} \text{υπο τους περιορισμους} \quad & y_i (w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N, \\ & \xi_i \geq 0, \quad i = 1, \dots, N. \end{aligned}$$

Τα ξ_i εκφραζουν τις παραβιασεις (σφαλματα). Η συναρτηση $\phi(x)$ ειναι η συναρτηση χαρτογραφησης (mapping) που παει τα δειγματα στον χωρο υψηλοτερης διαστασης. Η παραμετρος C ειναι ενας παραγοντας ποινης, που τιμωρει τα λαθη. Οσο πιο μεγαλη τιμη παιρνει, τοσο περισσοτερο τιμωρει τα λαθη, και ετσι τοσο πιο πολυ προσεγγιζουμε το hard margin. Στην ουσια προσπαθει να

βρει ολο και πιο αυστηρο margin, που συνηθως σημαινει οτι περισσοτερα δειγματα γινονται support vectors και γι'αυτο ο αλγοριθμος αργουσε οσο ανεβαζα το C. Αντιστροφα, οσο πιο μικρη ειναι, τοσο περισσοτερο επιτρεπει λαθη, που σημαινει οτι βρισκει καλυτερο margin, μεχρι ενα οριο φυσικα. Φτιαχνουμε τη Lagrangian:

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \alpha_i \left[y_i(w^\top x_i + b) - 1 + \xi_i \right] - \sum_{i=1}^N \mu_i \xi_i,$$

οπου $\alpha_i \geq 0$ και $\mu_i \geq 0$ πολλαπλασιαστες Lagrange. Πρεπει:

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^N \alpha_i y_i x_i,$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^N \alpha_i y_i = 0,$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - \mu_i = 0 \Rightarrow 0 \leq \alpha_i \leq C.$$

Αντικαθιστώντας τα παραπάνω στην \mathcal{L} , παίρνουμε το δυαδικό (dual) πρόβλημα:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j x_i^\top x_j$$

υπό τους περιορισμούς $0 \leq \alpha_i \leq C, \quad i = 1, \dots, N,$

$$\sum_{i=1}^N \alpha_i y_i = 0.$$

Από το $w = \sum_i \alpha_i y_i x_i$, η συνάρτηση αποφασής τελικά γίνεται:

$$f(x) = w^\top x + b = \sum_{i=1}^N \alpha_i y_i x_i^\top x + b.$$

Τα δειγματα x_i για τα οποια $\alpha_i > 0$ ειναι τα *support vectors*. Απο την παραπάνω μορφη της συνάρτησης αποφασής, βλέπουμε οτι

η κατηγοριοποίηση εξαρτάται μόνο από τα α_i που είναι μεγαλύτερα του μηδενός, **δηλαδή από τα support vectors**, και κατά συνέπεια από τα εσωτερικά τους γινόμενα. Σε μη γραμμικά διαχωρίσιμα dataset, κάνουμε τη χαρτογράφηση $x \mapsto \phi(x)$. Τότε το dual γίνεται:

$$\max_{\alpha} \left(\sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \phi^T(x_i) \phi(x_j) \right)$$

υπό τους περιορισμούς $0 \leq \alpha_i \leq C, \quad i = 1, \dots, N,$

$$\sum_{i=1}^N \alpha_i y_i = 0.$$

Ορίζοντας $\phi(x_i)^T \phi(x_j) = K(x_i, x_j)$, η συνάρτηση απόφασης τελικά γίνεται:

$$f(x) = \sum_{i=1}^N \alpha_i y_i K(x_i, x) + b,$$

και η προβλεψη:

$$\hat{y} = \text{sign}(f(x)).$$

Το $K(x_i, x)$ είναι το Kernel trick που κάνουμε, ώστε να αποφύγουμε τον υπολογισμό του εσωτερικού γινομένου $\phi(x_i)^T \phi(x_j)$ στις τεραστιές διαστάσεις που δουλεύουμε. Από θεωρία γνωρίζουμε ότι, για συγκεκριμένους Kernel, κάποια διάσταση και κάποια απεικόνιση $x \rightarrow \phi(x)$, τέτοια ώστε:

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

δηλαδή ο Kernel υπολογίζει ακριβώς το εσωτερικό γινόμενο των $\phi(x)$ και $\phi(z)$. Για παράδειγμα, για τη συνάρτηση χαρτογράφησης που είπαμε παραπάνω, το εσωτερικό γινόμενο δύο χαρτογραφημένων δειγμάτων είναι:

$$\phi(x)^T \phi(z) = x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 x_2 z_1 z_2 = (x^T z)^2.$$

Ο πολυωνυμικός Kernel:

$$K(x, z) = (x^T z)^2,$$

υπολογίζει ακριβώς το ίδιο εσωτερικο γινόμενο. Επομένως, αρκεί ο υπολογισμός της kernel συναρτησης $K(x, z)$.

Δε θελω να αναλυσω παραπανω τη θεωρια των SVM. Για reference ειδα τα παρακατω βιντεο:

- <https://www.youtube.com/watch?v=ny1jZ5A8jLA&t=1s>
- <https://www.youtube.com/watch?v=PwhiWxHK8o&t=161s>

Μέρος 1: Αναζήτηση Βέλτιστου SVM

Linear Kernel

Ο γραμμικός kernel ορίζεται ως:

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$$

Παράμετρος:

- C : Ποινή Λαθων

Μοντέλο	C	MaxIter	Train Acc	Val Acc
Linear 1	1	2000	0.3985	0.3927
Linear 2	10	2000	0.3982	0.3931
Linear 3	0.1	2000	0.3983	0.3932
Linear 4	100	5000	0.3983	0.3926

Παρατηρούμε ότι ο γραμμικός πυρήνας δεν αποδίδει καλά. Αυτό συμβαίνει γιατί δεν μεταφέρει τα δεδομένα σε υψηλότερη διάσταση,

και προσπαθει να τα διαχωρισει γραμμικα στον αρχικο χωρο, το οποιο στο δικο μου dataset ειναι πολυ δυσκολο, λογω πληθους δειγματος και διαστασεων.

Polynomial Kernel

Ο πολυωνυμικος Kernel οριζεται ως:

$$K(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^\top \mathbf{x}' + coef0)^d$$

Παράμετροι:

- d : βαθμος πολυωνυμου. Προφανως η διασταση που πηγαινω αυξανεται εκθετικα με το d , και κατα συνεπεια και η πολυπλοκοτητα, κατι που δεν ειναι απαραιτητα καλο. Γι'αυτο οπως θα δουμε, απο $d = 2 \rightarrow d = 3$ εχουμε βελτιωση, αλλα απο $d = 3 \rightarrow d = 4$ οχι.
- γ : κλιμακα
- $coef0$: σταθερα που προσθετει ορους χαμηλοτερης ταξης στον πυρηνα, για σταθεροτητα.

Δεν χρησιμοποιοισα γ , οποτε πηρε την default τιμη του στον αλγοριθμο.

Μοντέλο	Degree	C	coef0	Train Acc	Val Acc
Poly 1	2	1	0	0.5495	0.4621
Poly 2	3	1	0	0.6547	0.4653
Poly 3	4	1	0	0.6372	0.3823
Poly 4	3	10	0	0.9072	0.4756
Poly 5	3	100	0	0.9919	0.4561
Poly 6	3	40	0	0.9760	0.4658
Poly 7	3	40	1	0.9957	0.5201

Παρατηρούμε το overfitting με την αύξηση του C , λόγω αυτών που είπαμε παραπάνω (ψαχνει τελειο διαχωρισμο κλπ). Επίσης βλέπουμε στο τελευταίο μοντέλο ότι η σταθερά $coef0$ βοήθησε στη γενίκευση.

RBF Kernel

Ο RBF Kernel ορίζεται ως:

$$K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$$

Παραμετροί:

- C : ποινή σφαλμάτων
- γ : ευρος επιδρασης κάθε δειγματος

Θεωρητικά, ο Kernel αυτός είναι απείρης διαστάσης διότι αν αναπτύξουμε το εκθετικό σε σειρά Taylor, προκύπτει:

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \dots = \sum_{k=0}^{\infty} \frac{t^k}{k!}$$

$$K(x, z) = \sum_{k=0}^{\infty} \frac{(-\gamma \|x - z\|^2)^k}{k!}.$$

Επομένως, ο RBF είναι σαν πολυωνυμικός kernel απείρου βαθμού. Όπως γράφω και στο notebook, έκανα κι άλλες δοκιμές που δεν φαίνονται παρακάτω, πριν βρω το μοντέλο 14 (δεν τις έβαλα τυχαία δηλαδή). Όπου βλέπουμε $\gamma = scale$, υπολογίζει:

$$\gamma = \frac{1}{n_{\text{features}} \cdot \text{Var}(X)}$$

όπου:

- n_{features} : ο αριθμός των χαρακτηριστικών εισόδου
- $\text{Var}(X)$: η διακύμανση των δεδομένων εισόδου

Μοντέλο	C	γ	Train Acc	Val Acc
RBF 1	1	0.1	1.0000	0.1035
RBF 2	10	0.1	1.0000	0.1058
RBF 3	1	1	1.0000	0.1000
RBF 4	1	0.01	1.0000	0.1983
RBF 5	1	0.001	0.8686	0.5428
RBF 6	1	0.0001	0.5126	0.4808
RBF 7	10	0.001	0.9980	0.5551
RBF 8	5	0.001	0.9910	0.5562
RBF 9	1	0.005	0.9970	0.2932
RBF 10	10	0.0005	0.9714	0.5516
RBF 11	7	0.0007	0.9822	0.5575
RBF 12	1	0.004	0.9937	0.3481
RBF 13	1	0.0007	0.7905	0.5495
RBF 14	2.1	0.00065	0.8907	0.5600
RBF 15	1	scale	0.6592	0.5360
RBF 16	0.1	scale	0.4708	0.4450
RBF 17	10	scale	0.9398	0.5536
RBF 18	7	scale	0.9085	0.5559
RBF 19	5	scale	0.8718	0.5565
RBF 20	3	scale	0.8049	0.5569

Παρατηρούμε ότι ο RBF kernel δίνει τα καλύτερα αποτελέσματα. Τα μοντέλα RBF 14 , RBF 19 ήταν πολύ κοντά σε απόδοση, όμως προτιμήσα το 19 γιατί έτρεξα και τα 2 τρεις φορές με διαφορετικά seeds, και αυτό ήταν πιο συνεπές.

Sigmoid Kernel

$$K(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^\top \mathbf{x}' + coef0)$$

Μοντέλο	C	γ	Train Acc	Val Acc
SIGM 1	1	0.1	0.1382	0.1380
SIGM 2	1	scale	0.2285	0.2283
SIGM 3	1	scale	0.1366	0.1336
SIGM 4	10	scale	0.2287	0.2287

Όπως βλέπουμε αυτός ο Kernel δίνει πολύ κακά αποτελέσματα, και δε χρησιμοποιείται πλέον σήμερα.

Best SVM Models

Μοντέλο	C	Παράμετρος	Train Acc	Val Acc
LINEAR 3	0.1	--	0.3983	0.3932
POLY 7	40	degree = 3	0.9957	0.5201
RBF 19	5	$\gamma = scale$	0.8718	0.5565
SIGM 4	1	$\gamma = scale$	0.2287	0.2287

Τελικό SVM

Το καλύτερο SVM μοντέλο ήταν:

Μοντέλο	Train Acc	Val Acc	Test acc
RBF 19	0.8718	0.5565	0.5535

Μέρος 2: Σύγκριση με MLP

Το MLP αποτελείται από:

- Ένα κρυφό επίπεδο 2048 νευρώνων
- Batch Normalization
- Dropout
- Output layer 10 νευρώνων

Χρησιμοποιήθηκε hinge loss:

$$\mathcal{L} = \max(0, 1 - y \cdot f(x))$$

Το MLP είχε σχεδόν ίδια αποδοση με το SVM:

- Train accuracy ≈ 0.71
- Validation accuracy ≈ 0.57
- Test accuracy: **0.5475**

Μέρος 3: CNN + SVM

Στο μέρος 3, σκεφτήκα να παρω το εκπαιδευμένο CNN που είχα φτιάξει στην 1η εργασία, η δομή του οποίου φαίνεται στο notebook CNN+SVM στο κελί `cnn.summary()`, χρησιμοποιώντας τα ετοιμα βάρη του που είχα αποθηκεύσει (τα οποία βρίσκονται στο αρχείο `final-model6.h5`).

Συγκεκριμένα, αφαιρώ το τελευταίο `dense(10)` layer που στην ουσία η softmax βρίσκει τις πιθανότητες ώστε να γίνει αποφαση-προβλεψη της κατηγορίας, και να χρησιμοποιήσω το καλύτερο SVM που βρήκα ως κατηγοριοποιητή. Πήρα τις εξόδους του activation 4 με την εντολή `feature_model = tf.keras.Model(inputs=cnn.inputs, outputs=cnn.get_layer("activation 4").output)`, η οποία στην ουσία κάνει forward pass στο CNN και παίρνει τις εξόδους του activation 4 layer.

Απο κει και περα εκανα οτι κανω στο Μερους 1, απλα με εισοδο στο SVM την εξοδο του CNN. Οπως φανηκε, αυτο ειναι πολυ καλη τεχνικη γιατι το CNN δινει πολυ καλυτερη εξοδο απο τα raw-pixel-values που εδινα στο SVM αρχικα (εχουν περασει ολοκληρη επεξεργασια μεσα απο layers, εχουν εντοπιστει μοτιβα κλπ). Επισης, δεν χρειαστηκε PCA εδω, καθως οπως βλεπουμε και στη δομη του CNN, το activation 4 δινει μια εξοδο μονο 256 διαστασεων. Πειραματιστηκα με διαφορα SVM κι εδω, Τα αποτελεσματα ειναι τα εξης:

- Validation accuracy: **0.9262**
- Test accuracy: **0.8755**

Όπως παρατηρουμε, η αποδοση ειναι σημαντικα ανωτερη απο το απλο SVM.

Παρατηρήσεις

- Κατά την προεπεξεργασία εφαρμόστηκε τόσο κανονικοποίηση ($x' = x/255$) όσο και scaling. Ωστόσο, το scaling καθιστά την κανονικοποίηση περιττή:

$$\mu' = \mu/255, \quad \sigma' = \sigma/255 \Rightarrow z' = \frac{x' - \mu'}{\sigma'} = \frac{x - \mu}{\sigma}$$

Άρα το τελικό αποτέλεσμα παραμένει ίδιο.

- Υπαρχουν confusion matrixes σε ολα τα notebooks για τα τελικα μοντελα (best SVM, CNN+SVM, MLP), καθως επισης learning curve για τον MLP, και παραδειγματα ορθης και εσφαλμενης προβλεψης (στο CNN+SVM).
- Οπως ειπα και στην εισαγωγη, χρησιμοποιησα ετοιμη υλοποιηση SVM απο την cuml. Επειδη αυτη δουλευει με GPU εκανα μετατροπες με την np.asarray πριν βαλω τα δεδομενα

στο SVM. Ωστόσο, επειδή οι μετρικές μου (π.χ accuracy score) είναι από την sklearn, ξαναέκανα μετατροπή όπου χρειάστηκε, με την `cp.asnumpy`.

- Οι χρόνοι εκπαίδευσης προβλεψής είναι στο notebook στις εξόδους των κελιών.
- Στο CNN+SVM, για να φορτώσω το CNN μου, περνούσα το αρχείο `final_model6.h5` κάθε φορά που ήθελα να πειραξώ το notebook. Οπότε το προσθέτω και αυτό στον φακέλο με τα notebooks.