

Νευρωνικα Δικτυα - Εργασια 2

Ρούσος Σταμάτης

22 Ιανουαρίου 2026

Εισαγωγή

Σε αυτη την εργασια ασχολουμαστε με τα SVM (Support Vector Machines) ως κατηγοριοποιητες. Εσπασα την εργασια σε 3 μερη.

1. Στο 1o Μερος, εψαξα να βρω το καλυτερο SVM, κρινοντας με βαση την αποδοση εκπαιδευσης και γενικευσης (training/validation accuracy).
2. Στο 2o Μερος, παιρνω το καλυτερο SVM που βρηκα, και το συγκρινω με ενα MLP Neural Network.
3. Στο 3o Μερος, πηρα το καλυτερο CNN που υλοποιησα στην 1η εργασια, και αντι για το τελευταιο layer κατηγοριοποιησης που ειχε, χρησιμοποιησα το καλυτερο SVM μου ως κατηγοριοποιητη.

Προκειμενου να δοκιμασω πολλα μοντελα SVM με διαφορετικες τιμες παραμετρων, χρησιμοποιησα την cuml για το SVM, η οποια χρησιμοποιει GPU και ετσι οι υπολογισμοι γινονταν πολυ πιο γρηγορα (λογω παραλληλιας). Απλα για να το κανω αυτο, χρειαστηκε να μετατρεψω τα δεδομενα σε συγκεκριμενο ειδος array για την GPU, το οποιο εκανα με την cupy. Αυτα εγιναν με τη βοηθεια του ChatGPT και συναδελφου που ρωτησα.

Προεπεξεργασία Δεδομένων

Το dataset που χρησιμοποιησα ειναι το CIFAR-10 το οποιο αποτελειται συνολικα απο 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 $[R \quad G \quad B]$ τιμες καθενος απο τα 32×32 πιξελ μιας εικονας. Οι 10.000 εικονες ειναι ηδη διαχωρισμενες, αποτελουν το test set και χρησιμοποιούνται 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 ειναι ολες στην ιδια κλιμακα (διαστημα $[0,1]$).

Στη συνεχεια, εκανα Scaling, με την StandardScaler και τις εντολες fit,transform. Η fit, υπολογιζει το μεσο ορο και την τυπικη αποκλιση 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 διαστασεων.

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 αυτό το προβλημα γραμμικα, δηλαδη αναζητωντας μια δια-

χωριστική γραμμή η οποία να διαχωρίζει σωστά τα δειγματα των δυο κλασεων. Ομως, γενικα υπαρχουν απειρες τετοιες ευθειες (αν το dataset ειναι γραμμικα διαχωρισιμο).

Η βελτιστη διαχωριστικη ευθεια, δηλαδη η ευθειας που επιτρεπει την καλυτερη-ευκολοτερη ταξινομηση χωρις λαθη (πχ σημεια κοντα στο οριο ή που εχουν περασει λιγο το οριο αλλα ειναι πιο κοντα στην δικη τους κλαση), το SVM επιλεγει εκεινη τη γραμμη που μεγιστοποιει την ελαχιστη αποσταση ολων των σημειων από αυτην. Ισοδυναμα, αρκει να μεγιστοποιησουμε τις αποστασεις των κοντινοτερων σημειων των δυο κλασεων απο τη διαχωριστικη γραμμη. Τα σημεια αυτα ονομαζονται *support vectors* και ειναι τα μονα δειγματα που επηρεαζουν τη θεση της διαχωριστικης γραμμης.

Η αποσταση των support vectors απο τη διαχωριστικη γραμμη 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 διαχωριστικης γραμμης. Γι' αυτο το λογο, συνηθως χρησιμοποιουμε soft margin σε δεδομενα που ειναι δυσκολα διαχωρισιμα (πχ πολλες διαστασεις ή πολλα δειγματα κλπ). Το soft Margin επιτρεπει λαθη, προσθετοντας ενα παραγοντα χαλαρωσης. Αυτο βοηθαι στην ευρεση καλυτερης λυσης του προβληματος. Σε datasets που δεν ειναι γραμμικα διαχωρισιμα (οπως στο δικο μας), μεταφερουμε τα δεδομενα σε υψηλοτερη διασταση, με σκοπο να γινουν γραμμικα διαχωρισιμα. Αυτο το κανουμε με τη χρηση καποιας συναρτησης χαρτογραφησης $\phi(x)$ που παιρνει καθε δειγμα και το μετασχηματιζει σε ψηλοτερη διασταση. Για παραδειγμα για ενα σημειο $x = (x_1, x_2)$ 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) χρησιμοποιουμε 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:

$$f(x) = w^\top x + b,$$

και η προβλεψη:

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

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

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ \text{υπο τους περιορισμους} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N, \\ & \xi_i \geq 0, \quad i = 1, \dots, N. \end{aligned}$$

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 σημαίνει ότι περισσότερα δειγματα γινονται support vectors και γι'αυτο ο αλγορίθμος αργουνσε οσο ανεβαζα το C. Αντιστροφα, οσο πιο μικρη ειναι, τοσο περισσότερο επιτρεπει λαθη, που σημαίνει ότι βρισκει καλυτερο margin, μεχρι ενα οριο φυσικα. Φτιαχνουμε τη Lagrangian:

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

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

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

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

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

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

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

$$\text{υπό τους περιορισμούς } 0 \leq \alpha_i \leq C, \quad i = 1, \dots, N, \\ \sum_{i=1}^N \alpha_i y_i = 0.$$

Από το $w = \sum_i \alpha_i y_i x_i$, η συναρτηση αποφασης τελικα γινεται:

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

Τα δειγματα x_i για τα οποια $\alpha_i > 0$ ειναι τα *support vectors*. Απο την παραπανω μορφη της συναρτησης αποφασης, βλεπουμε ότι

η κατηγοριοποιηση εξαρταται μονο απο τα α_i που ειναι μεγαλυτερα του μηδενος, **δηλαδη απο τα support vectors**, και κατα συνεπεια απο τα εσωτερικα τους γινομενα. Σε μη γραμμικα διαχωρισμα dataset, κανουμε τη χαρτογραφηση $x \mapsto \phi(x)$. Τοτε το dual γινεται:

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

υπο τους περιορισμους $0 \leq \alpha_i \leq C, \quad i = 1, \dots, N,$

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

Οριζοντας $\phi(x_i)^T \phi(x_j) = K(x_i, x_j)$, η συναρτηση αποφασης τελικα γινεται:

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

και η προβλεψη:

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

Το $K(x_i, x)$ ειναι το Kernel trick που κανουμε, ωστε να αποφυγουμε τον υπολογισμο του εσωτερικου γινομενου $\phi(x_i)^T \phi(x_j)$ στις τεραστιες διαστασεις που δουλευουμε. Απο θεωρια γνωριζουμε οτι, για συγκεκριμενους Kernel, καποια διασταση και καποια απεικονιση $x \longrightarrow \phi(x)$, τετοια ωστε:

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

δηλαδη ο Kernel υπολογιζει ακριβως το εσωτερικο γινομενο των $\phi(x)$ και $\phi(z)$. Για παραδειγμα, για τη συνάρτηση χαρτογραφησης που ειπαμε παραπανω, το εσωτερικο γινομενο δυο χαρτογραφημενων δειγματων ειναι:

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

Ο πολυωνυμικός Kernel:

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

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

Δε θελω να αναλυσω παραπάνω τη θεωρία 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 : Ποινη 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 πυρηνας δεν αποδιδει καλα. Αυτο συμβαινει γιατι δεν μεταφερει τα δεδομενα σε υψηλοτερη διασταση,

και προσπαθεί να τα διαχωρισεί γραμμικά στον αρχικό χωρο, το οποιο στο δίκο μου dataset είναι πολύ δυσκολό, λογω πληθους δειγμάτων 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 ανξανεται 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 χρησιμοποιησα 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 αυξηση του C, λογω αυτων που ειπαμε παραπανω (ψαχνει τελειο διαχωρισμο κλπ). Επισης βλεπουμε στο τελευταιο μοντελο ότι η σταθερα coef0 βοηθησε στη γενικευση.

RBF Kernel

Ο RBF Kernel οριζεται ως:

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

Παραμετροι:

- C : ποινη σφαλματων
- γ : ευρος επιδρασης καθε δειγματος

Θεωρητικα, ο Kernel αυτος ειναι απειρης διαστασης διοτι αν αναπτυξουμε το εκθετικο σε σειρα Taylor, προκυπτει:

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

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

Επομενως, ο RBF ειναι σαν πολυωνυμικος kernel απειρου βαθμου. Οπως γραφω και στο notebook, εκανα κι αλλες δοκιμες που δεν φαινονται παρακατω, πριν βρω το μοντελο 14 (δεν τις εβαλα τυχαια δηλαδη). Οπου βλεπουμε $\gamma = scale$, υπολογιζει:

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

οπου:

- n_{features} : ο αριθμος των χαρακτηριστικων εισοδου
- $\text{Var}(X)$: η διακυμανση των δεδομενων εισοδου

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 δινεί τα καλυτερά αποτελεσματα. Τα μοντέλα RBF 14 , RBF 19 ηταν πολύ κοντά σε αποδοση, ομως προτιμήσα το 19 γιατί ετρεξα και τα 2 τρεις φορες με διαφορετικα seeds, και αυτο ηταν πιο συνεπες.

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

Οπως βλεπουμε αυτος ο Kernel δινει πολυ κακα αποτελεσματα, και δε χρησιμοποιειται πλεον σημερα.

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

To MLP αποτελείται από:

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

Χρησιμοποιηθήκε hinge loss:

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

To MLP ειχε σχεδον ιδια αποδοση με το SVM:

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

Μέρος 3: CNN + SVM

Στο μερος 3, σκεφτηκα να παρω το εκπαιδευμενο CNN που ειχα φτιαξει στην 1η εργασια, η δομη του οποιου φαινεται στο notebook CNN+SVM στο κελι cnn.summary(), χρησιμοποιωντας τα ετοιμα βαρη του που ειχα αποθηκευσει (τα οποια βρισκονται στο αρχειο final-model6.h5).

Συγκεκριμενα, αφαιρω το τελευταιο dense(10) layer που στην ουσια η softmax βρισκει τις πιθανοτητες ωστε να γινει αποφαση-προβλεψη της κατηγοριας, και να χρησιμοποιησω το καλυτερο SVM που βρηκα ως κατηγοριοποιητη. Πηρα τις εξοδους του activation 4 με την εντολη feature model = tf.keras.Model(inputs=cnn.inputs, outputs=cnn.get_layer("activation 4").output), η οποια στην ουσια κανει forward pass στο CNN και παιρνει τις εξοδους του activation 4 layer.

Απο κει και περα εκανα οτι κανω στο Μερος 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

- Κατα 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 cp.asarray πριν βαλω τα δεδομενα

στο SVM. Ωστόσο, επειδή οι μετρικές μου (π .χ accuracy score) είναι από την sklearn, ξαναεκάνα μετατροπή οπου χρειαστηκε, με την cp.asnumpy.

- Οι χρονοί εκπαίδευσης προβλεψης είναι στο notebook στις εξόδους των κελιών.
- Στο CNN+SVM, για να φορτωσω το CNN μου, περνουσα το αρχειο final model6.h5 καθε φορα που ηθελα να πειραξω το notebook. Οποτε το προσθετω και αυτο στον φακελο με τα notebooks.