Νευρωνικά Δίκτυα - Βαθιά Μάθηση

Δεύτερη Εργασία

Ευαγγελία Δημαρίδου

I. Εισαγωγή

Το θέμα της παρούσας εργασίας είναι να υλοποιήσουμε ένα Support Vector Machine το οποίο θα επιλύει ένα πρόβλημα κατηγοριοποίησης. Συγκεκριμένα αυτό θα χρησιμοποιηθεί για τον διαχωρισμό 2 κλάσεων της Cifar-10. Ο αλγόριθμος του Support Vector Machine, επιχειρεί να μεγιστοποιήσει την απόσταση μεταξύ της διαχωριστικής επιφάνειας 2 κλάσεων, διατηρώντας παράλληλα χαμηλό τον αριθμό των λανθασμένων κατηγοριοποιήσεων. Υπάρχουν διάφορα είδη SVM, γραμμικά και μη γραμμικά. Συγκεκριμένα χωρίζονται σε κατηγορίες ανάλογα με την συνάρτηση kernel που χρησιμοποιούν. Στις υλοποιήσεις που ακολουθούν, χρησιμοποιήθηκαν 3 είδη:

- Linear
- Polynomial
- Radial Basis Function

Ακολουθήθηκε υλοποίηση στην python, αρχικά ενός Linear SVM ακολουθώντας την λογική του Gradient Descent, το οποίο βρίσκεται στο αρχείο linear_svm_from_scratch.py. Ακόμη χρησιμοποιώντας συναρτήσεις τετραγωνικού προγραμματισμού υλοποιήθηκε και ένα SVM με μεταβλητή συνάρτηση kernel, το οποίο βρίσκεται στο αρχείο kernel_svm. Για λόγους σύγκρισης χρησιμοποιήσαμε και έτοιμη κλάση SVM της βιβλιοθήκης Sklearn και η υλοποίηση βρίσκεται στο αρχείο sklearn_svm.

II. Linear SVM

Η πιο απλή υλοποίηση ενός SVM είναι αυτή του γραμμικού. Στην περίπτωση αυτού, χρησιμοποιήθηκαν 2 προσεγγίσεις του στην υλοποίηση του στην Python.

A. Πρώτη Προσέγγιση

1) Θεωρητική Προσέγγιση

- Αρχικά αντιμετωπίστηκε ως πρόβλημα Gradient Descent, με συγκεκριμένη συνάρτηση Loss και Learning Rate. Αυτό είναι εφικτό καθώς το πρόβλημα μεγιστοποίησης του margin, αντιμετωπίζεται στην διάσταση που βρίσκονται ήδη τα δεδομένα. Η συνάρτηση Loss (hinge loss) είναι η εξής:

$$L = \lambda * \|\mathbf{w}\|^2 + max(0, 1 - y_i(w * x_i - b))$$

Σε περίπτωση σωστής κατηγοριοποίησης του εκάστοτε δείγματος η συνάρτηση Loss απλοποιείται σε:

$$L = \lambda * \|\mathbf{w}\|^2$$

- Η παράμετρος λάμδα, είναι regularization παράμετρος η οποία φροντίζει να "τιμωρούνται" τα μεγάλα βάρη. Έτσι μεγαλύτερο λάμδα οδηγεί σε μικρότερα βάρη και καλύτερη γενίκευση του μοντέλου.

- Παίρνοντας τις παραγώγους ως προς w και b, ενημερώνουμε τα βάρη και το bias σε κάθε εποχή και για κάθε δείγμα εκπαίδευσης, παίρνοντας batch size ίσο με 1. Πρακτικά είναι σαν να έχουμε ένα dense layer και χρησιμοποιούμε back propagation.

2) Ανάλυση Κώδικα

Όσον αφορά την υλοποίηση σε κώδικα, δημιουργήσαμε μία κλάση SVM.

- Στην συνάρτηση αρχικοποίησης, ορίζουμε τις εποχές, το learning rate, καθώς και την τιμή του λάμδα.

- Στην συνάρτηση fit() πραγματοποιούμε την εκπαίδευση του SVM. Συγκεκριμένα, για κάθε εποχή και για κάθε δείγμα, ανανεώνουμε τα βάρη και το bias λαμβάνοντας υπόψιν την κατηγοριοποίηση του εκάστοτε δείγματος.

- Η συνάρτηση gradient_descent, η οποία καλείται από την συνάρτηση fit(), πραγματοποιεί ουσιαστικά την ανανέωση των βαρών και του bias.

- Η συνάρτηση predict() πολλαπλασιάζει το κάθε δείγμα ελέγχου με τα βάρη, προσθέτει το bias, και ανάλογα με το πρόσημο του αποτελέσματος, κατηγοριοποιεί το εκάστοτε δείγμα.

- Η συνάρτηση accuracy() χρησιμοποιεί για κάθε δείγμα ελέγχου την παραπάνω συνάρτηση και συγκρίνοντας με την πραγματική κλάση (ground truth), υπολογίζει και εκτυπώνει το συνολικό accuracy.

B. Δεύτερη Προσέγγιση

Για την δημιουργία γραμμικού SVM, μπορούμε να εφαρμόσουμε την τεχνική που χρησιμοποιεί τετραγωνικό προγραμματισμό και συναρτήσεις kernel. Παρακάτω παρατίθεται ο τύπος της γραμμικής συνάρτησης Kernel:

$$K(x_i, x_j) = x_i \cdot x_j^T$$

III. Non Linear SVM

A. Θεωρητική Προσέγγιση

Στην περίπτωση ενός μη γραμμικού SVM, το όριο απόφασης δεν είναι γραμμικό, ή υπερεπίπεδο. Πρόκειται για μία πιο περίπλοκη επιφάνεια. Αυτό συμβαίνει γιατί οι ομοιότητες και οι διαφορές των δεδομένων, εξετάζονται σε ένα υψηλότερο επίπεδο, δηλαδή αυξάνεται η διάσταση των δεδομένων. Έτσι, βλέποντας πως συσχετίζονται τα δεδομένα σε υψηλότερη διάσταση, εφαρμόζουμε τα συμπεράσματα στην διάσταση που μας ενδιαφέρει. Βέβαια πράξεις με σημεία μεγάλης διάστασης θα είχαν υψηλό υπολογιστικό κόστος. Με την τεχνική όμως του kernel, δεν χρειάζεται να φέρουμε τα σημεία σε υψηλότερη διάσταση προκειμένου να κάνουμε τους απαραίτητους υπολογισμούς.

Στην προσέγγιση που ακολουθήθηκε, το ζητούμενο είναι να ελαχιστοποιήσουμε μία συνάρτηση και ταυτόχρονα να ισχύει μία ανισότητα. Ελαχιστοποιούμε την συνάρτηση:

$$\frac{1}{2} * w \cdot w^T + C * \sum_i \xi_i$$

ενώ θέλουμε να ισχύουν οι εξής προυποθέσεις:

$$y_i * (w^T \cdot \Phi(x_i) + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

Το πρόβλημα αυτό το λύνουμε χρησιμοποιώντας πολλαπλασιαστές Lagrange και τελικά ανάγεται στο εξής πρόβλημα μεγιστοποίησης:

$$-\frac{1}{2} * a^T \cdot H \cdot a + a^T * e$$

Παράλληλα θέλουμε να ισχύουν οι εξής προυποθέσεις

$$\sum_i a_i * y_i = 0$$

$$0 \leq a_i \leq C, \forall i$$

- Η παράμετρος C είναι penalty παράμετρος και "τιμωρεί" τις λανθασμένες κατηγοριοποίησεις. Όσο μεγαλύτερη είναι η τιμή της τόσο λιγότερα λάθη ανέχεται το SVM, όταν δημιουργεί την διαχωριστική υπερεπιφάνεια.

- Οι παράμετροι r και d χρησιμοποιούνται στην Polynomial Kernel συνάρτηση ως εξής:

  $$K(x_i, x_j) = (x_i \cdot x_j^T + r)^d$$

  Η παράμετρος d επηρεάζει την πολυπλοκότητα του SVM καθώς και το flexibility του ορίου απόφασης. Όταν d = 1, τότε ο Polynomial kernel θα συμπεριφέρεται ως Linear, παράγοντας ένα υπερεπίπεδο ως διαχωριστική επιφάνεια. Έτσι για μεγαλύτερο d, δεδομένα με πιο περίπλοκες σχέσεις, διαχειρίζονται καλύτερα. Ένα πολύ μεγάλο d μπορεί βέβαια να οδηγήσει και σε overfitting.

- Η παράμετρος γάμμα, χρησιμοποιείται στην RBF kernel συνάρτηση ως εξής:

  $$K(x_i, x_j) = e^{-gamma * \|\mathbf{x_i} - \mathbf{x_j}\|^2}$$

  Όπως φαίνεται από την παραπάνω εξίσωση, όσο αυξάνεται το γάμμα, τόσο μικραίνουν οι τιμές των kernel. Έτσι σημεία που απέχουν πολύ μεταξύ τους, επηρεάζουν όλο και λιγότερο το όριο απόφασης. Για το κάθε σημείο δηλαδή δίνεται περισσότερη έμφαση στα σημεία που βρίσκονται κοντά τους, και τις ομοιότητες του με αυτά. Γενικά μεγαλύτερες τιμές του γάμμα μπορούν να οδηγήσουν σε πιο περίπλοκο όριο απόφασης που θα είναι πιο ευαίσθητο σε overfitting.

Προκειμένου να λύσουμε το παραπάνω πρόβλημα τετραγωνικού προγραμματισμού καλούμε την συνάρτηση solvers.qp() της βιβλιοθήκης cvxopt.

Έτσι αφού βρήκαμε πώς θα λύσουμε το κύριο πρόβλημα του SVM στην Python, προχωράμε στην υλοποίηση του κώδικα.

B. Ανάλυση Κώδικα

Δημιουργούμε και σε αυτή την περίπτωση μία κλάση SVM.

- Φτιάχνουμε αρχικά την συνάρτηση αρχικοποίησης, η οποία παίρνει ως ορίσματα:
  - Το είδος kernel
  - Το r του Polynomial
  - Το degree του Polynomial
  - Το gamma του RBF
  - Την penalty παράμετρο C

- Η συνάρτηση compute_kernel(), παίρνει ως παραμέτρους 2 διανύσματα και υπολογίζει την συνάρτηση kernel για αυτά τα δείγματα, που είναι ένας αριθμός. Φυσικά ανάλογα με το είδος του kernel που έχει οριστεί, εφαρμόζει τον κατάλληλο τύπο.

- Η συνάρτηση construct_kernel_matrix() υπολογίζει τον πίνακα H με τους τετραγωνικούς συντελεστές του πίνακα με τα alpha. Ο τύπος του πίνακα αυτού είναι:

  $$H(k, j) = y_k * y_j * (\Phi({x_k}^T) \cdot \Phi(x_j))$$

  όπου y είναι οι κλάσεις των δειγμάτων και Φ η συνάρτηση Kernel. Καθώς αυτός ο πίνακας θα δοθεί ως όρισμα στην συνάρτηση solvers.qp(), πρέπει να είναι matrix όπως αυτό ορίζεται από την βιβλιοθήκη cvxopt και έτσι περνάει από την συνάρτηση αυτή.

- Στην συνάρτηση construct_qp_param(), υπολογίζονται τα υπόλοιπα ορίσματα της συνάρτησης solvers.qp()

  - Ο πίνακας linear_coef αποτελεί τον πίνακα e στον τύπο που παρουσιάσαμε πάνω, αλλά πολλαπλασιασμένο με το -1. Αυτό συμβαίνει γιατί η συνάρτηση που χρησιμοποιούμε, ελαχιστοποιεί μία συνάρτηση οπότε πολλαπλασιάζουμε την δικιά μας με το -1, καθώς θέλουμε να την μεγιστοποιήσουμε.

- Οι πίνακες A και b αναφέρονται στην πρώτη προυπόθεση ισότητας που παρουσιάσαμε, με τον πίνακα A να σχετίζεται με το αριστερό, ενώ ο πίνακας b με το δεξί μέρος της ισότητας.

- Οι πίνακες G και h, αναφέρονται στην προυπόθεση ανισότητας οι οποία μάλιστα έχει δύο μέρη. Για αυτό χρησιμοποιώντας τις συναρτήσεις np.vstack() και np.hstack() της numpy, δημιουργούμε τους κατάλληλους πίνακες.

- Τέλος ο πίνακας init_values περιέχει αρχικοποίηση των alphas, με τιμή 0.

- Στην συνάρτηση quadr_program() καλείται η συνάρτηση solvers.qp() και γίνεται επεξεργασία των αποτελεσμάτων της. Αρχικά βρίσκουμε τα alpha παίρνοντας την λύση solution['x'] και τα αποθηκεύουμε στον πίνακα self.all_alphas. Έπειτα βρίσκουμε ποια βρίσκονται στα όρια που θέλουμε δηλαδή είναι support vectors και έχουν

$$0 < a \leq C,$$

και αποθηκεύουμε τα αντίστοιχα indexes στον πίνακα self.is_sv. Ακολούθως αποθηκεύουμε όλα τα labels self.sv_y, τα alpha self.alphas, τα δείγματα self.sv και τα στοιχεία του πίνακα P self.quadr_coef_ind που αναφέρονται σε Support Vectors.

- Στην συνάρτηση compute_bias() υπολογίζουμε το bias του SVM. Για τον σκοπό αυτό παίρνουμε ένα support vector και εφαρμόζουμε το δείγμα αυτό και την κλάση του, στον τύπο:

$$b = y_s - w^T * \Phi(x_s)$$

- Στην συνάρτηση fit() πραγματοποιούμε την εκπαίδευση του SVM, καλώντας τις συναρτήσεις που αναφέραμε παραπάνω.

- Η συνάρτηση predict, παίρνει ως ορίσματα τα δείγματα ελέγχου και τις αντίστοιχες κλάσεις τους. Υπολογίζει την κλάση που προέβλεψε το SVM και αποθηκεύει το αποτέλεσμα στον πίνακα self.predicted_labels. Χρησιμοποιείται ο τύπος:

$$class = sign(w^T * x_k + b)$$

Βέβαια το w δίνεται από τον τύπο:

$$W = \sum_k a_k * y_k * \Phi(x_k)$$

Το άθροισμα αυτό αναφέρεται σε όλα τα support vectors και φυσικά ισχύει:

$$K(x, y) = \Phi(x) \cdot \Phi(y)$$

- Τα αποτελέσματα της παραπάνω συνάρτησης, επεξεργάζονται από την συνάρτηση accuracy(), η οποία εκτυπώνει το συνολικό accuracy, training ή test.

IV. Επεξεργασία Δεδομένων

A. Επιλογή Κλάσεων

Σε αυτή την εργασία, όπως και στην προηγούμενη επιλέχθηκε το σύνολο δεδομένων της Cifar-10. Συγκεκριμένα λόγω της φύσεως του SVM, και επειδή η διαδικασία multiclass classification θα καθιστούσε τα πειράματα ιδιαίτερα χρονοβόρα, κατηγοριοποιούμε μόνο 2 κλάσεις. Χρησιμοποιήσαμε 2 ζευγάρια κλάσεων:

- Αεροπλάνο και Φορτηγό, κλάσεις 0 και 9 αντίστοιχα. Οι κατηγορίες αυτές είναι αρκετά διαχωρίσιμες και έτσι αναμένουμε μεγάλο ποσοστό επιτυχίας σε αυτή την κατηγοριοποίηση.
- Γάτα και Σκύλος, κλάσεις 3 και 5 αντίστοιχα. Η επιλογή αυτή έγινε λόγω κάποιων παρατηρήσεων στην πρώτη εργασία. Στην κατηγοριοποίηση με την μέθοδο του back propagation και συγκεκριμένα χρησιμοποιώντας Multilayer Perceptron, έγινε φανερό ότι ο σκύλος συγχέεται με την γάτα. Οι περισσότερες λανθασμένες κατηγοριοποιήσεις οφειλόταν στις ομοιότητες αυτών των 2.

Έτσι προκειμένου να εξετάσουμε αυτά τα ζευγάρια, επεξεργαζόμαστε κατάλληλα τα δεδομένα εκπαίδευσης και ελέγχου. Ακόμη διαιρούμε τα δεδομένα με το 255, για λόγους numerical stability.

B. Επιλογή Παραμέτρου Γάμμα

Στα πειράματα μας, επιχειρήσαμε να βρούμε την βέλτιστη τιμή για την παράμετρο γάμμα του RBF. Καθώς δοκιμάσαμε και έτοιμες βιβλιοθήκες SVM, παρατηρήσαμε ότι υπάρχει η επιλογή "scale". Έτσι βρήκαμε έναν τύπο ο οποίος με βάση την διασπορά των δεδομένων βρίσκει ένα επαρκές γάμμα. Στην περίπτωση μας αυτό ήταν:

$$gamma = \frac{1}{n * variance} \approx 0.005$$

Εδώ το n είναι ο αριθμός των χαρακτηριστικών του κάθε δείγματος, δηλαδή 3072, και variance η απόκλιση όλου του συνόλου εκπαίδευσης.

V. Πειράματα

Προκειμένου να παρατηρήσουμε την συμπεριφορά του SVM με την αλλαγή των παραμέτρων του, πραγματοποιήσαμε μία σειρά από πειράματα.

A. *Linear*

Αρχικά χρησιμοποιήσαμε γραμμική συνάρτηση Kernel.

1) Κλάσεις *0* και *9*

Τιμή C	Training Accuracy	Test Accuracy
0.5	87.64 %	78.65 %
1	88.46 %	77.95 %
2	89.36 %	77.6 %
3	89.91 %	77.55 %
4	90.21 %	77.95 %
5	90.36 %	77.7 %

- Το καλύτερο Test Accuracy παρατηρείται για C = 0.5 και είναι 78.65 %.

2) Κλάσεις *3* και *5*

Τιμή C	Training Accuracy	Test Accuracy
0.5	71.14 %	58.7 %
1	77.95 %	58.7 %
2	72.69 %	58.2 %
3	75.01 %	57 %
4	75.36 %	56.85 %
5	75.84 %	56.7 %

- Το καλύτερο Test Accuracy παρατηρείται για C = 0.5 και 1 και είναι 58.7 %.

3) Συμπεράσματα

Όπως φαίνεται από τα παραπάνω αποτελέσματα, όταν αυξάνεται το C αυξάνεται το Training Accuracy αφού τιμωρούνται τα λάθη στην διάρκεια της εκπαίδευσης. Έτσι αφού αυτό οδηγεί σε πιο πολύπλοκα μοντέλα, παρατηρείται πτώση του testing accuracy. Ακόμη είναι φανερό ότι στις κλάσεις 0 και 9, οι οποίες είναι πιο διαχωρίσιμες, συναντάμε καλύτερα ποσοστά σωστής κατηγοριοποίησης. Τέλος συγκρίνοντας τα αποτελέσματα αυτά με το έτοιμο μοντέλο sklearn, παρατηρήσαμε ότι τα αποτελέσματα ήταν πάρα πολύ κοντά, έως πανομοιότυπα.

B. Linear With Gradient Descent

Πραγματοποιήσαμε 3 πειράματα για κάθε ζευγάρι κλάσεων, με μεταβλητό learning rate. Το λάμδα το κρατήσαμε σταθερό και ίσο με 10^{-3}. Ακόμη για Validation Set χρησιμοποιήσαμε το Test Set. Το Test Accuracy που φαίνεται είναι για την τελευταία εποχή του κάθε πειράματος.

1) Κλάσεις *0* και *9*

- Για Learning Rate 10^{-3}:

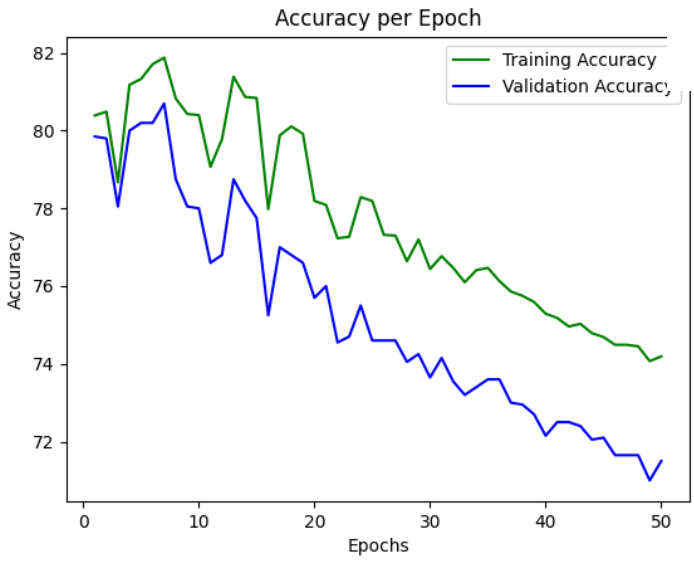


Figure 1: Testing Accuracy: 71.5 %

- Για Learning Rate 10^{-4}

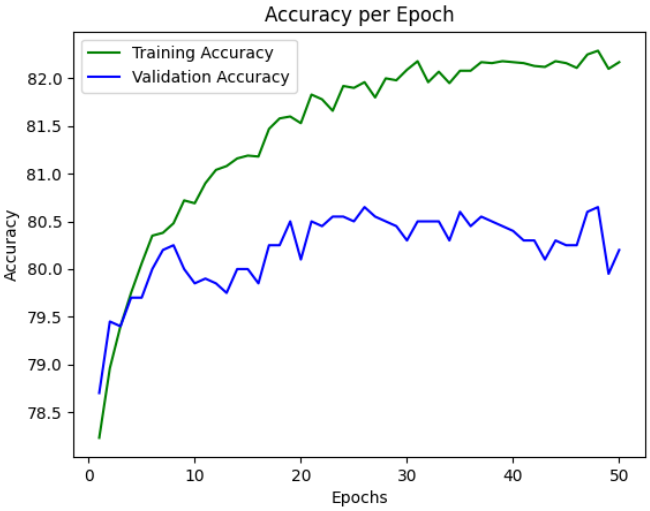


Figure 2: Testing Accuracy: 80.2 %

- Για Learning Rate 10^{-5}

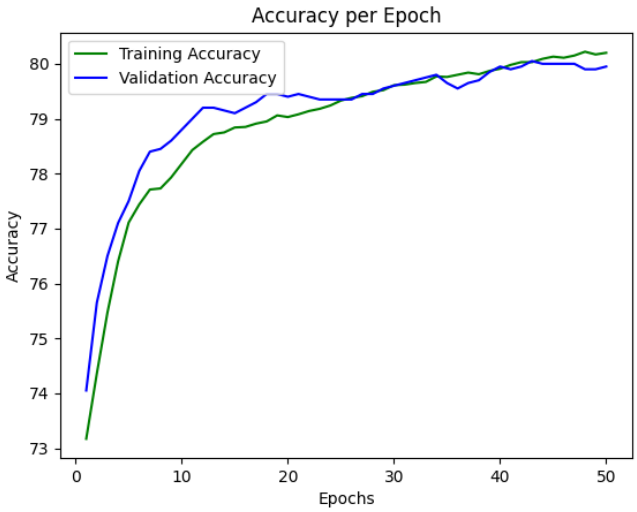


Figure 3: Testing Accuracy: 79.95 %

2) Κλάσεις *3* και *5*

- Για Learning Rate 10^{-3}:

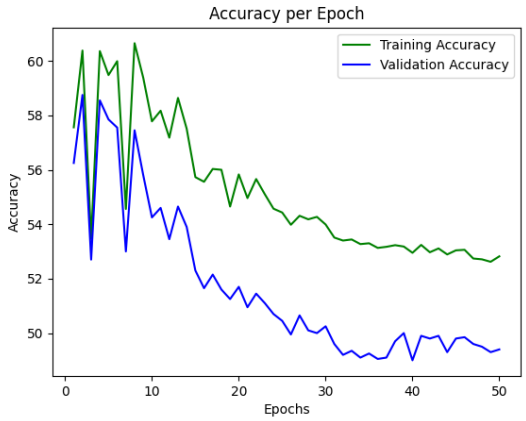


Figure 4: Testing Accuracy: 49.4 %

- Για Learning Rate 10^{-4}

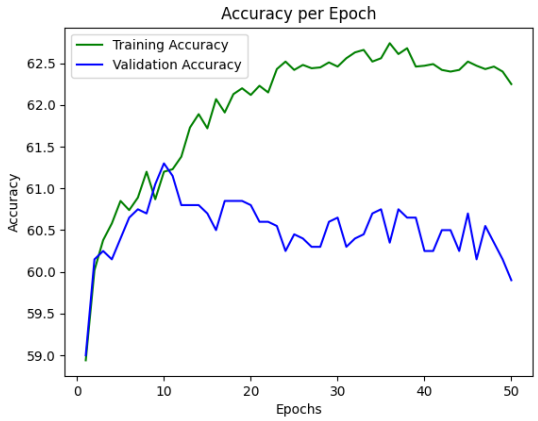


Figure 5: Testing Accuracy: 59.9 %

- Για Learning Rate 10^{-5}

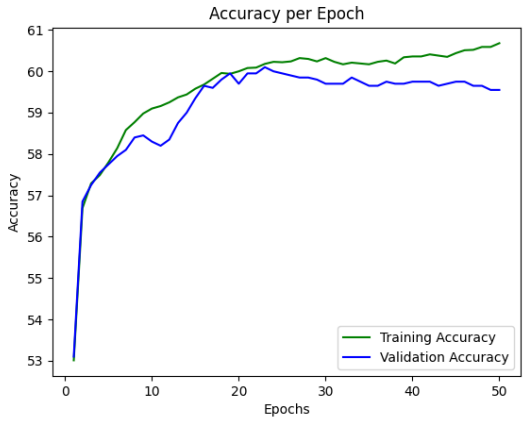


Figure 6: Testing Accuracy: 59.55 %

3) Παρατηρήσεις

- Όπως φαίνεται απο τα Test Accuracy των παραπάνω πειραμάτων, η μέθοδος που χρησιμοποιήθηκε οδήγησε σε ορθά αποτελέσματα. Η χρήση δηλαδή Gradient Descent, δίνει περίπου ίδια ποσοστά με την χρήση Linear Kernel και συνάρτηση τετραγωνικού προγραμματισμού.

- Όσον αφορά τις διαφορετικές τιμές learning rate, γίνεται φανερό ότι μικρότερη τιμή learning rate οδηγεί σε πιο ομαλή σύγκλιση του Validation Accuracy, καθώς και σε λιγότερο overfitting.

- Ο χρόνος εκτέλεσης των πειραμάτων ήταν σημαντικά πιο μικρός σε σχέση με Linear Kernel, με αυτόν να μειώνεται όσο μειώνουμε το learning rate.

C. RBF

Για τα πειράματα με RBF kernel, μεταβλητές ήταν οι παράμετροι gamma και C. Τα πειράματα έγιναν και με έτοιμη βιβλιοθήκη SVM Sklearn και αναφέρονται ενδεικτικά κάποιες τιμές για λόγους σύγκρισης.

1) Κλάσεις *0* και *9*

- Για gamma 0.001:

Τιμή C	Training Accuracy	Test Accuracy
0.5	83.67 %	84.05 %
1	86.69 %	85.7 %
2	88.51 %	85.95 %
3	90.35 %	86.25 %
4	90.35 %	86.25 %
5	90.88 %	86.15 %
10	92.3 %	85.35 %

Sklearn με C = 1:

- Training Accuracy = 87.02 %
- Test Accuracy = 85.85%

- Για gamma 0.005:

Τιμή C	Training Accuracy	Test Accuracy
0.5	91.31 %	87.7 %
1	93.96 %	88.4 %
2	96.96 %	88.8 %
3	98.22 %	88.95 %
4	98.96 %	88.7 %
5	99.37%	88.75 %

Sklearn με C = 0.5:

- Training Accuracy = 91.31 %
- Test Accuracy = 87.7%

- Για gamma 0.01:

Τιμή C	Training Accuracy	Test Accuracy
0.5	93.76 %	87.4 %
1	97.62 %	88.35 %
2	99.53 %	89.35 %
3	99.76 %	89.3 %
4	99.87 %	89.15 %
5	99.94 %	89.2 %
10	100 %	89.4 %

  Sklearn με C = 3:
  - Training Accuracy = 99.76 %
  - Test Accuracy = 89.3%

- Για gamma 0.1:

Τιμή C	Training Accuracy	Test Accuracy
0.5	79.65 %	51.2 %
1	100 %	61.85 %
2	100 %	64.2 %
3	100 %	64.2 %
4	100 %	64.2 %
5	100 %	64.2 %
10	100 %	64.2 %

  Sklearn με C = 5:
  - Training Accuracy = 100 %
  - Test Accuracy = 64.2%

- Τα καλύτερα ποσοστά παρατηρούνται για gamma 0.01. Συγκεκριμένα για γαμμα 0.01 και C 10 έχουμε Test Accuracy 89.4 %. Ακόμη όσο αυξάνεται το C αυξάνεται πάλι και το training accuracy. Για μεγάλη τιμή του γάμμα παρατηρούμε μεγάλο overfitting αφού η απόκλιση μεταξύ training και testing accuracy για γαμμα 0.1, είναι η μεγαλύτερη. Το μοντέλο δηλαδή είναι πιο περίπλοκο και δεν γενικεύεται σωστά στα δείγματα ελέγχου.

2) Κλάσεις *3* και *5*

- Για gamma 0.001:

Τιμή C	Training Accuracy	Test Accuracy
0.5	61.71 %	61.2 %
1	63.19 %	61.65 %
2	66.98 %	62.95 %
3	70.77%	65.1 %
4	72.5 %	64.85 %
5	74.27 %	65.2 %

  Sklearn με C = 1:
  - Training Accuracy = 66.94 %
  - Test Accuracy = 64%

- Για gamma 0.005:

Τιμή C	Training Accuracy	Test Accuracy
0.5	73.94 %	65.95%
1	81.94 %	66.55 %
2	89.95 %	67.85 %
3	94.02 %	67.85%
4	96.05 %	68.05 %
5	97.26 %	68.1 %

  Sklearn με C = 1:
  - Training Accuracy = 81.94 %
  - Test Accuracy = 66.55%

- Για gamma 0.01:

Τιμή C	Training Accuracy	Test Accuracy
0.5	83.24 %	66.15 %
1	92.59 %	68.35 %
2	98.2 %	68.45 %
3	99.38 %	68 %
4	99.87 %	67.65 %
5	99.94 %	67.7 %
10	99.98 %	66.9 %

  Sklearn με C = 3:
  - Training Accuracy = 99.38 %
  - Test Accuracy = 68%

- Για gamma 0.1:

Τιμή C	Training Accuracy	Test Accuracy
0.5	88.7 %	50 %
1	100 %	55.3 %
2	100 %	54.5 %
3	100 %	54.5 %
4	100 %	54.5 %
5	100 %	54.5 %
10	100 %	54.5 %

  Sklearn με C = 4:
  - Training Accuracy = 100 %
  - Test Accuracy = 54.45%

- Στις κλάσεις 3 και 5 παρατηρούμε πάλι ότι τα καλύτερα αποτελέσματα δίνονται για γαμμα 0.01. Μάλιστα για C = 2 έχουμε το καλύτερο αποτέλεσμα απο όλα τα RBF πειράματα, με testing accuracy 68.45 %. Όσο αυξάνεται το C παρατηρούμε ότι τα ποσοστά παραμένουν ίδια. Αυτό μπορεί να σημαίνει ότι πλέον δεν γίνονται άλλα λάθη ώστε αυτά να τιμωρηθούν αφού βλέπουμε και training accuracy 100 %.

D. Polynomial

Στα επόμενα πειράματα χρησιμοποιήσαμε Polynomial Kernel συνάρτηση, μεταβάλλοντας τις παραμέτρους degree και C.Για λόγους σύγκρισης παρατίθενται και κάποια αποτελέσματα με την έτοιμη βιβλιοθήκη.

1) Κλάσεις *0* και *9*

- Για degree 2:

Τιμή C	Training Accuracy	Test Accuracy
0.5	73.15 %	72.65 %
1	73.15 %	72.65 %
2	73.2 %	72.65 %
3	73.15 %	72.65 %
4	73.15 %	72.65 %
5	73.15 %	72.65 %

  Sklearn με C = 1:
  - Training Accuracy = 91.64 %
  - Test Accuracy = 86.15%

- Για degree 3:

Τιμή C	Training Accuracy	Test Accuracy
0.5	76.49 %	75.6 %
1	76.48 %	75.6 %
2	76.48 %	75.6 %
3	76.49 %	75.6 %
4	76.49 %	75.6 %
5	76.5 %	75.6 %
10	76.5%	75.6 %

  Sklearn με C = 1:
  - Training Accuracy = 99.78 %
  - Test Accuracy = 86.45%

- Για degree 4:

Τιμή C	Training Accuracy	Test Accuracy
0.5	78.8 %	76.9 %
1	63.58 %	64.2%
2	77.81 %	75.7 %
3	77.96 %	75.8 %
4	77.83 %	75.8 %
5	78.12 %	76.05 %
10	77.37 %	75.25%

  Sklearn με C = 1:
  - Training Accuracy = 100 %
  - Test Accuracy = 86%

- Για degree 5:

Τιμή C	Training Accuracy	Test Accuracy
0.5	63.97 %	64.1%
1	63.82 %	64.15%
2	63.21 %	63.75 %
3	68.42 %	66.85 %
4	67.98%	66.5 %
5	67.35 %	66.15 %
10	67.7%	66.25%

  Sklearn με C = 1:
  - Training Accuracy = 100 %
  - Test Accuracy = 86%

Παρατηρούμε ότι για polynomial kernel έχουμε χειρότερα αποτελέσματα απότι για RBF. Συγκεκριμένα ενώ για RBF πετύχαμε αποτέλεσμα 89.4 %, εδώ το μέγιστο ήταν 76.9 % για degree = 4 και C = 0.5. Το γεγονός ότι πετύχαμε καλύτερο πσοοστό για degree = 4 δείχνει ότι αύξηση του degree μπορεί να βελτιώσει τα αποτελέσματα αλλά υπερβολική αύξηση του μπορεί και να τα χειροτερεύσει. Ακόμη παρατηρούμε ότι τα αποτελέσματα της βιβλιοθήκης Sklearn έχουν πολύ καλύτερο Training Accuracy και Test Accuracy.

2) Κλάσεις *3* και *5*

- Για degree 2:

Τιμή C	Training Accuracy	Test Accuracy
0.5	100 %	62.15 %
1	100 %	62.15 %
2	100 %	62.15 %
3	100 %	62.15 %
4	100 %	62.15 %
5	100 %	62.15 %
10	100 %	62.15 %

  Sklearn με C = 1:
  - Training Accuracy = 81.35 %
  - Test Accuracy = 65.3%

- Για degree 3:

Τιμή C	Training Accuracy	Test Accuracy
0.5	100 %	61.8 %
1	100 %	61.8 %
2	100 %	61.8 %
3	100 %	61.8 %
4	100 %	61.8 %
5	100 %	61.8 %
10	100 %	61.8 %

  Sklearn με C = 1:
  - Training Accuracy = 98.42 %
  - Test Accuracy = 64.7%

- Για degree 4:

Τιμή C	Training Accuracy	Test Accuracy
0.5	100 %	61.3 %
1	100 %	61.05 %
2	100 %	61.05 %
3	100 %	61.05 %
4	100 %	61.35 %
5	100 %	61.25 %
10	100 %	61.25 %

  Sklearn με C = 1:
  - Training Accuracy = 99.99 %
  - Test Accuracy = 62%

- Για degree 5:

Τιμή C	Training Accuracy	Test Accuracy
0.5	55.32 %	53.7 %
1	50.84 %	50.55%
2	92.88 %	61.05 %
3	91.36 %	61.4 %
4	92.29 %	61.5 %
5	90.52 %	61.45 %
10	90.95%	61.5%

  Sklearn με C = 1:
  - Training Accuracy = 100 %
  - Test Accuracy = 62.7%

- Παρατηρούμε ότι στις κλάσεις 3 και 5 υπάρχει περισσότερο overfitting και λιγότερο Test Accuracy. Ενώ το δίκτυο μαθαίνει καλύτερα τα δεδομένα εκπαίδευσης, δεν γενικεύεται σωστά σε όλα τα δείγματα ελέγχου. Το καλύτερο Test Accuracy ήταν για degree = 2 με C = 1 με test accuracy = 62.15 %.

- Ακόμη γίνεται φανερό ότι η βιβλιοθήκη Sklearn πετυχαίνει καλύτερο Test Accuracy, ενώ το Training Accuracy φαίνεται να είναι ίδιο στα περισσότερα πειράματα.

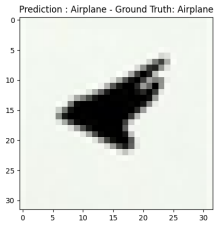
VI. Χαρακτηριστικά Παραδείγματα Ορθής και Εσφαλμένης Κατηγοριοποίησης

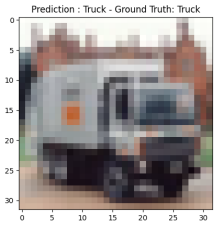
A. Κλάσεις *0* και *9*

Προκειμένου να παρουσιάσουμε κάποιες εικόνες που κατηγοριοποιήθηκαν λανθασμένα και μη, επιλέγουμε το πείραμα με το καλύτερο Test Accuracy. Αυτό ήταν: RBF με Gamma = 0.01 και C = 10, με Test Accuracy 89.4%

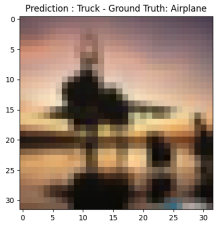
Κλάση	Testing Accuracy
0	88.4 %
9	90.4 %

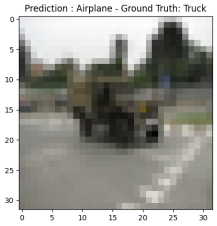
Φωτογραφίες σωστών κατηγοριοποιήσεων:





Φωτογραφίες λανθασμένων κατηγοριοποιήσεων:





Όπως φαίνεται από τις παραπάνω εικόνες, αυτές που κατηγοριοποιήθηκν σωστά είναι απολύτως διαχωρίσιμες με το ανθρώπινο μάτι. Οι εικόνες που κατηγοριοποιήθηκαν λανθασμένα μπορούν να χαρακτηριστούν πιο ”δύσκολες”.

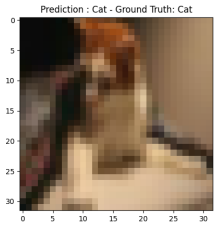
B. Κλάσεις *3* και *5*

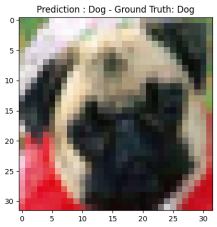
Για τις κλάσεις αυτές, το πείραμα με το καλύτερο Test Accuracy ήταν αυτό με RBF Kernel, Gamma = 0.01 και C = 2, με Test Acuracy 68.45%

Κλάση	Testing Accuracy
3	69 %
5	67.9 %

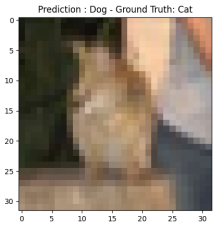
Παρατηρούμε ότι η γάτα είχε μεγαλύτερο ποσοστό σωστής κατηγοριοποίησης από τον σκύλο, αν και η διαφορά είναι μικρή.

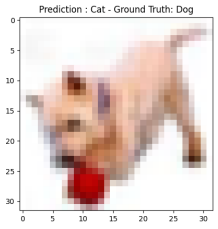
Φωτογραφίες σωστών κατηγοριοποίησεων:





Φωτογραφίες λανθασμένων κατηγοριοποίησεων:





Παρατηρούμε ότι στις φωτογραφίες που κατηγοριοποιήθηκαν σωστά, με το ανθρώπινο μάτι καταλαβαίνουμε πολύ εύκολα την σωστή κατηγορία. Στις λανθασμένες, και συγκεκριμένα στην πρώτη, δεν είναι τόσο ξεκάθαρο ότι απεικονίζεται γάτα.

VII. Χρόνοι Εκπαίδευσης

- Καθώς τρέχαμε ταυτόχρονα πολλά πειράματα, δεν μετρήθηκε ακριβής χρόνος για όλα. Ένας ενδεικτικός χρόνος για τα πειράματα, βρίσκεται στον παρακάτω πίνακα:

Kernel	Sklearn	From Scratch
Linear	4 mins	20 mins
RBF	10 mins	25 mins
Polynomial	12 mins	22 mins

- Παρατηρήσαμε ότι όσο αυξανουμε το C, τόσο αυξανεται και ο χρόνος εκπαίδευσης.
- Για τα Polynomial η αύξηση του degree δεν επηρέασε τον χρόνο εκπαίδευσης.
- Για αυξημένο gamma, επίσης αυξάνεται ο χρόνος εκπαίδευσης.
- Για το γραμμικό SVM με gradient descent, οι χρόνοι εκπαίδευσης ήταν κάτω από ένα λεπτό.

VIII. Σύγκριση SVM με KNN και Nearest Centroid

Οπως και στην προηγούμενη εργασία έτσι και σε αυτή, συγκρίνουμε την υλοποίηση μας με 2 αλγόριθμους. Αυτοί είναι οι κατηγοριοποιητές Πλησιέστερου Γείτονα και Πλησιέστερου Κέντρου.

A. Κλάσεις *0* και *9*

Πείραμα	Test Accuracy
Linear C = 0.5	78.65 %
Linear Gradient Descent	80.2%
RBF Gamma = 0.01 και C = 10	89.4%
Polynomial Degree = 4, C = 0.5	76.9 %
KNN K = 1	71.2 %
KNN K = 3	69.6 %
Nearest Centroid	73.65 %

B. Κλάσεις *3* και *5*

Πείραμα	Test Accuracy
Linear C = 1	58.7 %
Linear Gradient Descent	60%
RBF Gamma = 0.01 και C = 2	68.45%
Polynomial Degree = 2, C = 1	62.15 %
KNN K = 1	57.5 %
KNN K = 3	56.9 %
Nearest Centroid	57.95 %

C. Συμπεράσματα

- Παρατηρούμε ότι το SVM πετυχαίνει πολύ καλύτερα αποτελέσματα στην κατηγοριοποίηση των 2 κλάσεων. Η πολυπλοκότητα του SVM, και το γεγονός ότι αποτελεί ένα πρόβλημα βελτιστοποίησης υπό συνθήκες, συμβάλλουν στα καλύτερα αποτελέσματα του.
- Το RBF δίνει τα δεύτερα καλύτερα αποτελέσματα και στα δύο ζευγάρια κλάσεων.
- Στον KNN είναι πιο αποδοτικό το k = 1 δηλαδή το να υπολογίζεται μόνο η κλάση στην οποία ανήκει ο πλησιέστερος γείτονας στην απόφαση.
- Ο Nearest Centroid αποφέρει καλύτερα αποτελέσματα πό τον KNN αλλά χειρότερα από όλα τα καλύτερα ποσοστά των SVM.
- Η υλοποίηση του SVM με linear kernel, για δεδομένα που έχουν μεγάλες διαστάσεις, όπως τα δικά μας, παρουσιάζει και αυτή καλή αποτελέσματα. Μάλιστα για τις κλάσεις 0 και 9, καλύτερα ακόμη και από το Polynomial Kernel.

IX. Βιβλιογραφία

- https://scikit-learn.org/stable/
- https://en.wikipedia.org/wiki/Support_vector_machine
- https://towardsdatascience.com
- https://www.geeksforgeeks.org/support-vector-machine-algorithm/
- https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
- https://data-flair.training/blogs/svm-kernel-functions/