2η Εργασία στα Νευρωνικά Δίκτυα – SVM

Εργασία του Ανδρέα Σεγκάνι ΑΕΜ 10770

I. ΕΙΣΑΓΩΓΗ

Η συγκεκριμένη εργασία αποσκοπεί στην ανάπτυξη και την αξιολόγηση του αλγορίθμου των Support Vector Machines (SVM) για τη κατηγοριοποίηση των δεδομένων της βάσης CIFAR-10. Ο αλγόριθμος SVM αποδεικνύεται στη συνέχεια ότι είναι ισχυρότερος από τις μεθόδους που χρησιμοποιήσαμε σε προηγούμενες εργασίες , καθώς είναι σε θέση να διαχωρίζει δεδομένα ακόμη και όταν δεν είναι γραμμικά διαχωρίσιμα , χρησιμοποιώντας τη μέθοδο των πυρήνων (kernels).

Στη συνέχεια της εργασίας, θα εξεταστεί ο τρόπος με τον οποίο λειτουργεί ο αλγόριθμος που αναπτύξαμε στο Google Colab και θα εξετάσουμε τα προβλήματα διαχωρισμού κλάσεων. Θα χρησιμοποιηθούν πυρήνες οι οποίοι είναι γραμμικοί (linear) και μη (rbf , polynomial) για τη βελτιστοποίηση του μοντέλου και τη βελτίωση της ακρίβειας της κατηγοριοποίησης.

Τέλος, για την ανάλυση των αποτελεσμάτων θα εξεταστεί η αποτελεσματικότητα του SVM σε σύγκριση με άλλες μεθόδους κατηγοριοποίησης , τόσο σε όρους ακρίβειας όσο και σε υπολογιστική απόδοση. Επιπλέον, θα διερευνηθούν οι παράμετροι εκπαίδευσης , όπως και ο ρυθμός εκμάθησης , η επιλογή του πυρήνα και οι υπερπαράμετροι , οι οποίοι επηρεάζουν την απόδοση του αλγορίθμου.

II. Επεξήγηση του αλγορίθμου SVM

Το SVM βασίζεται στην ιδέα της εύρεσης ενός υπερεπίπεδου το οποίο διαχωρίζει τα δεδομένα διαφορετικών κατηγοριών με τον καλύτερο δυνατό τρόπο καθώς μεγιστοποιεί το περιθώριο (margin), δηλαδή την απόσταση μεταξύ του υπερεπιπέδου και των πλησιέστερων σημείων των κατηγοριών (support vectors).

Το υπερεπίπεδο ορίζεται ως το άθροισμα του γινομένου των βαρών w και του διανύσματος x των δεδομένων που έχουμε , με το bias. Δηλαδή w*x+b . Το υπερεπίπεδο σχηματίζει δύο κατηγορίες με αυτόν τον τρόπο , τη κατηγορία +1 και τη κατηγορία -1. Εάν w*x+b >=1 ανήκει στη κατηγορία +1 και αν w*x+b<=-1 ανήκει στη κατηγορία -1 αντίστοιχα. Οι συνθήκες αυτές μπορούν να γραφούν και ως

$$y_i(w * x_i + b) \geq 1 \text{ όπου } yi \in \{-1,+1\}.$$

- Η κατεύθυνση του w καθορίζει τον προσανατολισμό του υπερεπιπέδου.
- Το bias μετατοπίζει το επίπεδο

Για γραμμικά SVM μπορούμε να πούμε ότι το για ένα 2D χώρο το υπερεπίπεδο(το οποίο είναι μια γραμμή) έχει εξίσωση $\beta_0 + \beta_1 * \chi 1 + \beta 2 * \chi 2 = 0$

Τα σημεία πάνω από τη γραμμή αυτή θα ικανοποιούν τη παρακάτω συνθήκη :

$$\beta_0 + \beta_1 * \chi 1 + \beta 2 * \chi 2 > 0$$

Τα σημεία πάνω από τη γραμμή αυτή θα ικανοποιούν τη παρακάτω συνθήκη :

$$\beta_0 + \beta_1 * \chi 1 + \beta 2 * \chi 2 < 0$$

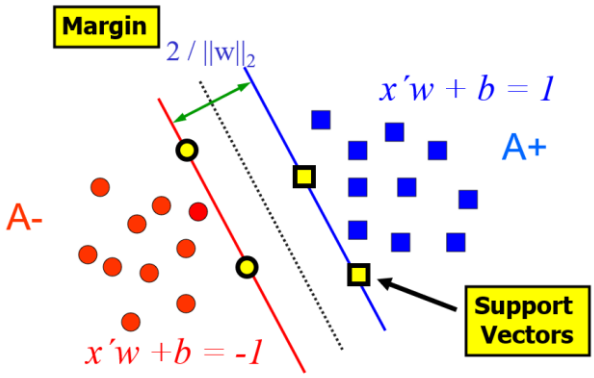
Και αν θέσουμε yi ∈ {−1,+1} όπου -1 αφορά ,για παράδειγμα τις κόκκινες κουκκίδες , και +1 αφορά τις μπλε , μπορούμε να ενώσουμε τις δύο σχέσεις σε μια:

$$yi(\beta_0 + \beta_1 * \chi 1 + \beta 2 * \chi 2) > 0$$

Γενικότερα, σκοπός του SVM είναι να διαχωρίζει τις δύο κατηγορίες με το να βρίσκει εκείνες τις παραμέτρους w και b που μεγιστοποιούν την ελαχιστοποίηση του ‖w‖ , έτσι ώστε να μεγιστοποιηθεί το περιθώριο (margin) του διαχωρισμού .

$$\text{Margin} = 2 / \|w\|$$

Το υπερεπίπεδο με μέγιστο margin είναι εκείνο που εξασφαλίζει τη καλύτερη δυνατή απόσταση από το πλησιέστερο σημείο δεδομένων κάθε κλάσης.



Έχουμε δύο ειδών margins: Το hard margin και το soft margin.

i. Το hard margin εφαρμόζεται όταν τα δεδομένα είναι γραμμικά διαχωρίσιμα, δηλαδή όταν υπάρχει ένα υπερεπίπεδο που μπορεί να χωρίσει τα δεδομένα χωρίς λάθη. Στη δική μας περίπτωση του CIFAR-10 τέτοια δεδομένα αναπαριστούν η σύγκριση της κλάσης 0 (αεροπλάνο) και 3 (γάτα) , όπου οι διαφορές είναι πολύ μεγάλες , πράγμα που σημαίνει τα δεδομένα είναι ξέχωρα μεταξύ τους. Η εξίσωση για το υπερεπίπεδο είναι $yi*(w^Tx_i +b)\geq 1$. Αν όλα τα δεδομένα ικανοποιούν τη παραπάνω ισότητα, τότε το περιθώριο είναι σκληρό.

ii. Το soft margin εφαρμόζεται όταν τα δεδομένα **δεν** είναι γραμμικά διαχωρίσιμα. Το soft margin επιτρέπει σε κάποια σημεία να βρίσκονται εντός του περιθωρίου ή στη λάθος πλευρά του υπερεπιπέδου. Για να αντιμετωπιστούν αυτές οι παραβιάσεις, εισάγονται μεταβλητές χαλάρωσης ξi (slack variables) όπου προκύπτουν από τη hingeloss(εξηγείται παρακάτω). Για το soft margin η συνθήκη διαχωρισμού γίνεται:

$yi*(w^Tx_i +b)\geq 1-\xi i$

Έτσι θα έχουμε και μια συνάρτηση που βελτιστοποιεί το SVM με το soft margin :

$$\min_{w,b}(1/2 \ *\| \mathbf{w} \|^2 + C \sum_{i=1}^{N} \xi i)$$

$1/2 * \| w \|^2$ = *όρος μεγιστοποίησης του περιθωρίου C: Παράγοντας ελέγχου της ποινής για τα σφάλματα (υπερπαράμετρος)*

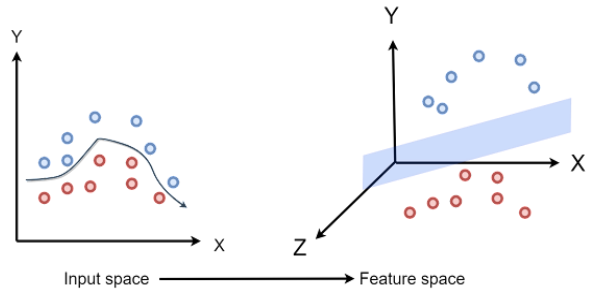
Έτσι σε περίπτωση που τα δεδομένα δεν είναι τέλεια διαχωρίσιμα, οι τιμές ξi>0 επιτρέπουν τη σωστή λειτουργία του . Οι παράμετροι αυτοί , όπως θα δούμε και στον αλγόριθμό μας υπολογίζονται μέσα από αυτόν , ενώ ο χρήστης μπορεί να δώσει τιμή στη παράμετρο C.

Επιπλέον, τα kernels είναι μία μαθηματική συνάρτηση που χρησιμοποιείται για να μετασχηματίσει τα δεδομένα από το αρχικό χαρακτηριστικό χώρο σε έναν υψηλότερο, ώστε να καταστεί δυνατό να βρεθεί ένα υπερεπίπεδο που να διαχωρίζει τα δεδομένα, ακόμα και αν αυτά δεν είναι γραμμικά διαχωρίσιμα στον αρχικό χώρο.

Τύποι kernels :

1. Linear kernel. Ο πιο απλός τύπος kernel, που χρησιμοποιείται όταν τα δεδομένα είναι ήδη γραμμικά διαχωρίσιμα (τύπου αεροπλάνο-σκύλος). Η συνάρτηση kernel είναι απλώς το εσωτερικό γινόμενο των δύο διανυσμάτων: $K(x,y)=x^Ty$, και χρησιμοποιείται όταν τα δεδομένα δεν απαιτούν μετασχηματισμό σε υψηλότερη διάσταση. Όταν ο πυρήνας είναι γραμμικός η απόφαση του υπερεπιπέδου υπολογίζεται από την εξίσωση **$ti = yi*(w^Tx_i +b)$.**

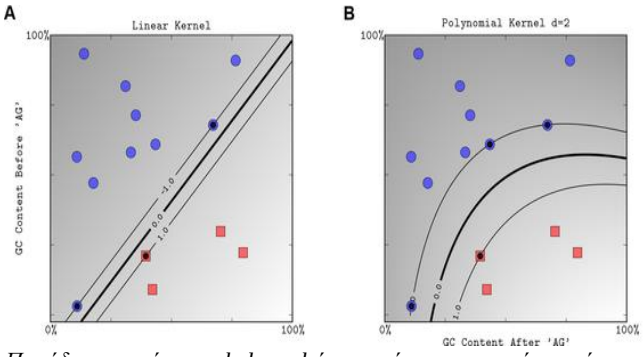
2. RBF kernel. RBF kernel είναι ένας **μη γραμμικός πυρήνας**, και χρησιμοποιείται για να μεταφέρει τα δεδομένα σε έναν υψηλότερο χώρο χαρακτηριστικών όπου τα δεδομένα μπορεί να είναι γραμμικά διαχωρίσιμα, ακόμα κι αν στον αρχικό χώρο χαρακτηριστικών δεν είναι. Για τον **RBF πυρήνα**, χρησιμοποιείται η έκφραση: $K(x,y) = e^{-\gamma norm(x-y)\wedge 2}$ όπου γ είναι η παράμετρος που ρυθμίζει την εξάπλωση του πυρήνα. Ο πυρήνας εδώ χρησιμοποιείται στον υπολογισμό των προβλέψεων του μοντέλου και του υπερεπιπέδου, και έτσι επηρεάζει έμμεσα την απώλεια. Για έναν RBF πυρήνα η απόφαση υπολογίζεται ως
$ti = yi *(\sum_{j=1}^{N} exp(-\gamma\|x_i - x_j\|^2) + b)$ (1) , το οποίο μετά χρησιμοποιείται για τη συνάρτηση hingeloss=max(0,1-ti). Αν το ti είναι μεγαλύτερο από 1, η απώλεια είναι 0 (δηλαδή, το δείγμα έχει ταξινομηθεί σωστά). Αν το ti είναι μικρότερο από 1, η απώλεια είναι μεγαλύτερη, καθώς το δείγμα είτε είναι λάθος ταξινομημένο ή βρίσκεται πολύ κοντά στην απόφαση του υπερεπιπέδου.



Παράδειγμα της συνάρτησης πυρήνα RBF που χαρτογραφεί δεδομένα από μη γραμμικά διαχωρίσιμο χώρο σε υψηλής διάστασης διαχωρίσιμο χώρο.

3. Polynomial kernel. είναι ιδανικός για περιπτώσεις όπου τα δεδομένα είναι μη γραμμικά διαχωρίσιμα, αλλά η σχέση μεταξύ των χαρακτηριστικών των δεδομένων μπορεί να περιγραφεί με έναν πολυωνυμικό τύπο. Δεδομένα σε δύο διαστάσεις x1 και x2,ο πολυωνυμικός πυρήνας υπολογίζει το εσωτερικό γινόμενο τους και στη συνέχεια το ανυψώνει σε μια υψηλότερη διάσταση χρησιμοποιώντας μια δύναμη, π.χ., (x1·x2+1)^d. Είναι λιγότερο περίπλοκος από τον RBF πυρήνα και παρέχει μια καλή εναλλακτική για δεδομένα με πιο απλές μη γραμμικές σχέσεις. Για έναν poly πυρήνα η απόφαση υπολογίζεται ως

$$\mathbf{ti = yi *}(\sum_{j=1}^{N}(x_i^T * x_j + 1)^{degree} + b) \quad (2)$$



Παράδειγμα χρήσης poly kernel όπου φαίνεται και η σύγκρισή του με το linear

III. *Επεξήγηση Αλγορίθμου*

Αρχικά στον αλγόριθμο φορτώνω το dataset CIFAR-10, δημιουργώ πίνακες για τα δεδομένα και τις ετικέτες και μετασχηματίζουμε τις εικόνες σε δισδιάστους πίνακες. Το SVM που εκπαιδεύτηκε έκανε διαχωρισμό δύο κλάσεων. Έτσι για παράδειγμα επέλεξα 1 ζευγάρι κλάσεων όπου τα χαρακτηριστικά τους είναι γραμμικά διαχωρίσιμα(αεροπλάνο-γάτα) και ένα ζευγάρι όπου τα χαρακτηριστικά τους δεν είναι γραμμικά διαχωρίσιμα(γάτα – σκύλος) μιας και παρουσιάζουν αρκετές ομοιότητες. Επιπλέον, λόγω των υψηλών υπολογιστικών απαιτήσεων και απαίτηση πολύ χρόνου για τη διαδικασία της εκπαίδευσης ,περιόρισα το μέγεθος του συνόλου εκπαίδευσης στα 1000 δείγματα. Εάν δεν ελαττώσουμε τα δεδομένα και δεν εφαρμόσουμε PCA στις διαστάσεις των δεδομένων ,τότε η διάσταση εισόδου Xtrain είναι 50.000 X 3072 , πράγμα που δικαιολογεί τη ραγδαία αύξηση του χρόνου εκπαίδευσης. Για αυτόν τον λόγο επέλεξα για training 1000 δείγματα ενώ για testing 200 δείγματα. Έπειτα γίνεται «marking» για τα δεδομένα(πχ γάτα-> 0 , σκύλος->1) καθώς και μετατροπή των ετικετών τους σε -1 και 1

ΑΛΓΟΡΙΘΜΟΣ SVM

```
class SVM:
    def __init__(self, C, kernel, degree, gamma):
        self.C = C
        self.w = 0
        self.b = 0
        self.kernel = kernel
        self.degree = degree
        self.gamma = gamma

    def kernel_function(self, x1, x2):
        if self.kernel == 'linear':
            return np.dot(x1, x2)
        elif self.kernel == 'poly':
            return (np.dot(x1, x2) + 1) ** self.degree
        elif self.kernel == 'rbf':
          if self.gamma == 'scale':
                self.gamma = 1 / x1.shape[0]
          return np.exp(-self.gamma * np.linalg.norm(x1 - x2) ** 2)
        else:
            raise ValueError("Λάθος μέθοδος")
```

Αρχικά, ορίζονται οι βασικές παράμετροι του SVM , όπως τον παράγοντα C (trade-off parameter), το kernel(πχ linear , polynomial, rbf) , ο βαθμός degree του πολυωνύμου σε περίπτωση όπου kernel = poly, το gamma (παράμετρος εξάπλωσης του πυρήνα στα rbf) καθώς και τα bias και τα weights.

Kernel_function: Υλοποιεί τις μεθόδους kernel που χρησιμοποιήθηκαν στην εργασία με βάση τα προαναφερθέντα.

```python
def hingeloss(self, w, b, X, Y):
    reg = 0.5 * np.dot(w, w.T)  #  1/2 * norm (w)

    hinge_losses = 0
    for i in range(X.shape[0]):
        opt_term = Y[i] * (np.dot(w, X[i]) + b)  # y*(w*x + b)
        hinge_losses += max(0, 1 - opt_term)

    total_loss = reg + self.C * hinge_losses  # Ολική απώλεια (κανονικοποίηση + συνολική απώλεια hinge)

    return total_loss
```

Hingeloss: συνάρτηση που υπολογίζει το σφάλμα της πρόβλεψης του μοντέλου και οδηγεί στην εκπαίδευση για να βελτιώσει τη γενίκευση. Μαθηματικά για τη hingeloss ισχύει το εξής:

L(y,f(x)) = max(0,1-y*f(x)) , όπου y η πραγματική κλάση (-1 ή 1) και f(x)(=w*x+b) η έξοδος του για το συγκεκριμένο datapoint. Εάν η πρόβλεψη είναι σωστή τότε y*f(x) ≥1 (μακριά και από το όριο) και τότε L =0 (δεν υπάρχει απώλεια). Όταν το αποτέλεσμα y*f(x) είναι μεταξύ 0 και 1 αν και το ταξινομεί σωστά , δίνουμε ένα penalty όπου υπολογίζεται από την L , δείχνοντας στο μοντέλο ότι πρέπει να αναπροσαρμόσει τα βάρη έτσι ώστε να αυξήσει την απόσταση από το όριο. Τέλος εάν y*f(x) τότε L>0 το μοντέλο πρέπει να αναπροσαρμόσει τα βάρη ώστε να διορθώσει τη λάθος ταξινόμηση.

Με βάση τον τύπο

$$\mathbf{1/2} \ *\| \mathbf{w} \|^2 + \boldsymbol{C} \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{\xi i}$$

Υπολογίζουμε το total_loss.

```python
def fit(self, X, Y, batch_size, learning_rate, epochs):
    if isinstance(X, torch.Tensor):
        X = X.cpu().numpy()
    if isinstance(Y, torch.Tensor):
        Y = Y.cpu().numpy()
    self.learning_rate = learning_rate
    self.epochs = epochs
    num_samples = X.shape[0]
    num_features = X.shape[1]
    c = self.C
    ids = np.arange(num_samples)
    np.random.shuffle(ids)
    w = np.zeros((1, num_features))
    b = 0
    losses = []
    for epoch in range(epochs):
        l = self.hingeloss(w, b, X, Y)
        losses.append(l)
        for batch_start in range(0, num_samples, batch_size):
            gradw = np.zeros((1, num_features))
            gradb = 0
            for j in range(batch_start, min(batch_start + batch_size, num_samples)):
                x = ids[j]
                if self.kernel == 'linear':
                    ti = Y[x] * (np.dot(w, X[x].reshape(1, -1).astype(np.float64).T) + b)
                else:
                    kernel_sum = np.sum([self.kernel_function(X[x], X[i]) for i in range(num_samples)])
                    ti = Y[x] * (kernel_sum + b)
                if np.any(ti > 1):
                    gradw += 0
                    gradb += 0
                else:
                    gradw += (c * Y[x] * X[x].reshape(1, -1))
                    gradb += c * Y[x]
            gradw = gradw.astype(np.float64)
            w = w - learning_rate * w + learning_rate * gradw  #
            b = b + learning_rate * gradb

        self.w = w
        self.b = b
```

Fit: Πρωτού αναλυθεί ο αλγόριθμος , πρέπει να επισημανθεί ότι για την ελαχιστοποίηση της συνάρτησης απώλειας χρησιμοποιήθηκε η μέθοδος της Gradient Descent, όπου υπολογίζονται οι παράγωγοι των βαρών και του bias για . Σε αυτό το μέρος του κώδικα βρίσκεται η κύρια διαδικασία της εκπαίδευσης. Αρχικά Τα βάρη w και το bias αρχικοποιούνται σε 0. Έπειτα , για κάθε εποχή υπολογίζεται η συνολική απώλεια με τη συνάρτηση hingeloss και αποθηκεύει τη τιμή της σε μια λίστα με όνομα losses , την οποία αρχικοποιήσαμε λίγο πιο πάνω. Τα δεδομένα χωρίζονται σε batches και αρχικοποιούνται τα gradient των βαρών w ίσο με 0.

Η επεξεργασία του κάθε δείγματος σε κάθε batch αρχικά , πραγματοποιείται με τον έλεγχο του kernel που χρησιμοποιείται. Εάν ο πυρήνας είναι γραμμικός τότε υπολογίζεται απευθείας το γινόμενο y * f(x) . Εάν δεν είναι γραμμικός , υπολογίζει το kernel_sum ,δηλαδή το άθροισμα που διατυπώθηκε στις σχέσεις (1) και (2) στη προηγούμενη σελίδα. Στη συνέχεια εάν ti>1 σημαίνει ότι το margin είναι μεγαλύτερο από 1 και επομένως έχει ταξινομηθεί σωστά οπότε τα βάρη θα μείνουν ως έχουν. Εάν ti≤ 1 (δηλαδή όταν βρίσκεται στο όριο ή έχει ταξινομηθεί λάθος) τότε υπολογίζονται τα grads και ανανεώνονται τα βάρη σύμφωνα με το gradient descent.

Μετά τον αλγόριθμο SVM είναι σημαντικό να επισημανθεί ότι έγινε χρήση PCA κρατώντας το 90% των χαρακτηριστικών επειδή χωρίς αυτό , ο αλγόριθμος καθυστερούσε σημαντικά

IV. Αποτελέσματα

Εκτός από τον αλγόριθμο που κατασκευάστηκε από την αρχή έγινε χρήση και ενός έτοιμου αλγορίθμου svm για λόγους σύγκρισης εγκυρότητας με τον αυτοσχέδιο. Αρχικά χρησιμοποίησα κλάσεις οι οποίες δεν είναι γραμμικά διαχωρίσιμες όπως για παράδειγμα τις κλάσεις 1 και 9 όπου αναπαριστούν αυτοκίνητο και φορτηγό αντίστοιχα . Διατηρώντας το learning rate ίσο με τη μονάδα για να μην έχουμε κάποια επιπλέον μεταβολή , τα αποτελέσματα για αναλυτική παραμετρική ανάλυση είναι τα εξής:

	RBF	LINEAR	POLY
C =0.1 , learning rate = 1	gamma=scale Training Accuracy: 65.40% Test Accuracy: 64.00%	Training Accuracy: 36.90% Test Accuracy: 34.50%	d=0.6 Training Accuracy: 61.60% Test Accuracy: 61.50%
	gamma=0.01 Training Accuracy: 63.30% Test Accuracy: 68.50%	---	d=1 Training Accuracy: 53.80% Test Accuracy: 54.00%
	gamma=0.1 Training Accuracy: 66.20% Test Accuracy: 68.00%	---	d=3 Training Accuracy: 51.70% Test Accuracy: 49.00%
	gamma=0.5 Training Accuracy: 62.50% Test Accuracy: 66.50%	--	d=4 Training Accuracy: 47.80% Test Accuracy: 45.50%
	gamma=1 Training Accuracy: 65.30% Test Accuracy: 66.00%	--	---
C = 0.5	gamma=scale Training Accuracy: 63.90% Test Accuracy: 68.00%	Linear SVM για C=0.5 XWRIS PCA Training Accuracy: 35.70% Test Accuracy: 33.50%	d=0.6 Training Accuracy: 63.60% Test Accuracy: 65.00%
	gamma=0.01 Training Accuracy: 64.40% Test Accuracy: 65.50%	---	d=1 Training Accuracy: 57.80% Test Accuracy: 55.00%
	gamma=0.5 Training Accuracy: 54.30% Test Accuracy: 51.00%	---	d=2 Training Accuracy: 47.80% Test Accuracy: 45.50%
C=1	gamma=scale Training Accuracy: 63.60% Test Accuracy: 68.00%	Training Accuracy: 36.20% Test Accuracy: 33.50%	d=0.6 Training Accuracy: 62.90% Test Accuracy: 66.50%
	gamma=0.01 Training Accuracy: 64.40% Test Accuracy: 62.50%	--	d=1 Training Accuracy: 62.10% Test Accuracy: 63.00%
	gamma=0.1 Training Accuracy: 63.10% Test Accuracy: 65.00%	--	d=2 Training Accuracy: 47.80% Test Accuracy: 45.50%
	gamma=0.5 Training Accuracy: 66.30% Test Accuracy: 65.00%	--	--
	gamma=1 Training Accuracy: 62.90% Test Accuracy: 65.50%	--	--
C=5	gamma=scale Training Accuracy:	Training Accuracy: 34.90% Test	--

	66.00% Test Accuracy: 67.50%	Accuracy: 34.00%	
	gamma=0.1 Training Accuracy: 55.70% Test Accuracy: 59.50%		--

Παρατηρούμε ότι οι πυρήνες rbf και poly έχουν εμφανώς καλύτερα αποτελέσματα μιας και τα δείγματα δεν είναι γραμμικά .

Στη περίπτωση όπου το γ = scale διασφαλίζεται ότι το γ προσαρμόζεται δυναμικά στο μέγεθος των δεδομένων, μειώνοντας τον κίνδυνο υπερπροσαρμογής ή υποπροσαρμογής , όπως φαίνεται και στον constructor innit.

Παρακάτω είναι οι σημαντικές παρατηρήσεις:

1. **Linear SVM:**
   - Οι χαμηλές τιμές C (C=0.1,0.5) οδηγούν σε πολύ χαμηλή ακρίβεια (Training: ~35%, Test: ~33%), κάτι που υποδεικνύει ότι το μοντέλο δεν καταφέρνει να διαχωρίσει καλά τα δεδομένα.
   - Το learning_rate είναι και αυτό αυξημένο πράγμα που σημαίνει απότομη αλλαγή των βαρών και να μη βελτιστοποιείται αφού βασίζεται μόνο στο gradient.
2. **Polynomial SVM:**
   - Χαμηλές τιμές του d (d=0.1,0.6) προσφέρουν καλύτερη ακρίβεια, με C=1 και d=0.1 να επιτυγχάνουν το καλύτερο αποτέλεσμα (Training: 65.50%, Test: 69.00%). Ωστόσο, καθώς αυξάνεται ο βαθμός (d≥2d), η απόδοση πέφτει δραματικά, κάτι που δείχνει υπερπροσαρμογή λόγω της πολυπλοκότητας.
   - Οι πολύ μεγάλες τιμές του d οδηγούν σε φτωχή γενίκευση, πιθανώς λόγω αριθμητικής αστάθειας.
3. **RBF SVM:**
   - Η επιλογή γ='scale' δίνει σταθερά καλά αποτελέσματα για διαφορετικές τιμές του C, με ακρίβεια Training ~65% και Test ~67-68%.
   - Πολύ μικρές τιμές γ (γ=0.01,0.1) δίνουν καλύτερη γενίκευση για C=0.1 και C=1, ενώ μεγαλύτερες τιμές γ (γ=5,10) οδηγούν σε υπερπροσαρμογή, μειώνοντας την απόδοση σε ορισμένα σενάρια.

Από τον έτοιμο αλγόριθμο που χρησιμοποιήθηκε είχα τα εξής αποτελέσματα για το rbf

SVM για C=0.1
gamma=scale, Train Accuracy: 74.95 , Test Accuracy: 72.50
SVM για C=0.5
gamma=scale, Train Accuracy: 84.64 , Test Accuracy: 77.90
SVM για C=1
gamma=scale, Train Accuracy: 88.77 , Test Accuracy: 78.55
SVM για C=5
gamma=scale, Train Accuracy: 97.80 , Test Accuracy: 80.30
SVM για C=10
gamma=scale, Train Accuracy: 99.49 , Test Accuracy: 80.20

Παρατηρούμε ότι υπάρχει διαφορά με τον δικό μου , αλλά για σταθερό γ καθώς αυξάνεται η παράμετρος C αυξάνεται και το accuracy έως ότου φτάσει σε μεγάλη τιμή (C=10). Ο χρόνος εκτέλεσης του αυτοσχέδιου SVM και για τους 3 πυρήνες με παραμετρική ανάλυση ήταν 6 ώρες και 14 λεπτά ενώ λόγω υπολογιστικών απαιτήσεων έπαψε να λειτουργεί στον poly στο σημείο όπου C=5.

Τα αποτελέσματα σε αυτοσχέδιο MLP το οποίο χρησιμοποιεί ένα μόνο κρυφό επίπεδο με 64 νευρώνες , και αξιοποιεί τη hinge loss για βελτιστοποίηση είχαμε αποτέλεσμα 47.44% και λόγω του ότι χρησιμοποιήθηκαν μόνο αυτές οι δύο κλάσεις , χρειάστηκαν περίπου δύο λεπτά για να τρέξει. Αυτό δεν είναι αρκετά καλά συγκρίσιμο επειδή και ο αλγόριθμος MLP είναι αυτοσχέδιος. Επιπλέον ο svm χρησιμοποιεί λιγότερα δεδομένα πράγμα που σημαίνει καλύτερη γενίκευση.

Επιπλέον τώρα για learning rate 0.001 έχουμε τα εξης αποτελέσματα για rbf

C=0.5	γ=scale	Training Accuracy: 75.50%	Test Accuracy: 78.50%
	gamma=0.1	Training Accuracy: 76.00%	Test Accuracy: 75.50%
	gamma=1	Training Accuracy: 75.50%	Test Accuracy: 75.50%
C=1	gamma=scale	Training Accuracy: 75.60%	Test Accuracy: 78.00%
	gamma=0.1	Training Accuracy: 76.20%	Test Accuracy: 75.50%
	gamma=1	Training Accuracy: 76.10%	Test Accuracy: 75.50%

Εδώ γενικά βλέπουμε λίγο καλύτερη επίδοση . Αυτό ίσως γίνεται λόγω του παραμέτρου learning rate το οποίο ελαττώθηκε και δεν επηρεάζει τόσο πολύ τα weights. Η καλύτερη απόδοση από εκείνη του SVC οφείλεται μάλλον στη μείωση της διάστασης μέσω pca που αφαιρεί θόρυβο και καθιστά τα δεδομένα λίγο πιο γραμμικά διαχωρίσιμα. Παρ'όλα αυτά η εφαρμογή του pca δεν είναι καλή τεχνική καθώς το μοντέλο δεν εκπαιδεύεται πλήρως.

Τέλος τα αποτελέσματα του KNN με 1 και 3 γείτονες για τις 2

αυτές κλάσεις είναι:

```
Accuracy με 1 γείτονα - Train: 100.00% Test: 68.60%
Accuracy με 3 γείτονες - Train: 85.89% Test: 70.20%
```

```
NCC - Train Accuracy: 63.43% Test Accuracy: 64.15%
```

Παρατηρούμε ότι η απόδοση του ncc είναι αρκετά μειωμένη.

Εάν επιλέξουμε τις κλάσεις 1 και 2 (αυτοκίνητο και πουλί) τότε για linear kernel σχεδόν για κάθε χαμηλή τιμή C έχουμε το εξής αποτέλεσμα :

```
print(f"Αριθμός σωστών ταξινομήσεων: {len(correct_idx)}")
print(f"Αριθμός λανθασμένων ταξινομήσεων: {len(incorrect_idx)}")
```

```
Αριθμός σωστών ταξινομήσεων: 158
Αριθμός λανθασμένων ταξινομήσεων: 42
```

Ενώ για C=1 και γ=scale έχουμε αυτά τα αποτελέσματα

```
Αριθμός σωστών ταξινομήσεων: 156
Αριθμός λανθασμένων ταξινομήσεων: 44
```

Και για poly kernel με d=2:

```
Αριθμός σωστών ταξινομήσεων: 152
Αριθμός λανθασμένων ταξινομήσεων: 48
```

Πράγμα που αντιστοιχεί σε ένα ποσοστό περίπου 76%.

Γενικότερα, επειδή τα δεδομένα είναι πιο εύκολα να διαχωριστούν το linear method είναι προτιμότερο , επειδή τρέχει σχετικά γρήγορα (περίπου 5 λεπτά για όλες τις επαναλήψεις) και βγάζει αν όχι καλύτερα , σχετικά ίδια αποτελέσματα με την rbf και poly.

Επίσης το accuracy σε αυτή τη περίπτωση φαίνεται να είναι καλύτερο από τους knn και ncc αφού

```
Accuracy με 1 γείτονα - Train: 100.00% Test: 74.40%
Accuracy με 3 γείτονες - Train: 80.24% Test: 70.75%
```

```
NCC - Train Accuracy: 74.16% Test Accuracy: 74.15%
```

Τέλος για την απόδοση του mlp στη περίπτωση αυτή έχουμε 48.61% απόδοση.

Γενικότερα παρατηρούμε ότι για learning rule 0.001 έχουμε τις καλύτερες αποδόσεις σε παραμετρικές αναλύσεις για το SVM . Όπως είδαμε και από τις συγκρίσεις γενικότερα προέκυψε ότι όσον αφορά το accuracy SVM > KNN & NCC > MLP(με ένα επίπεδο μόνο)