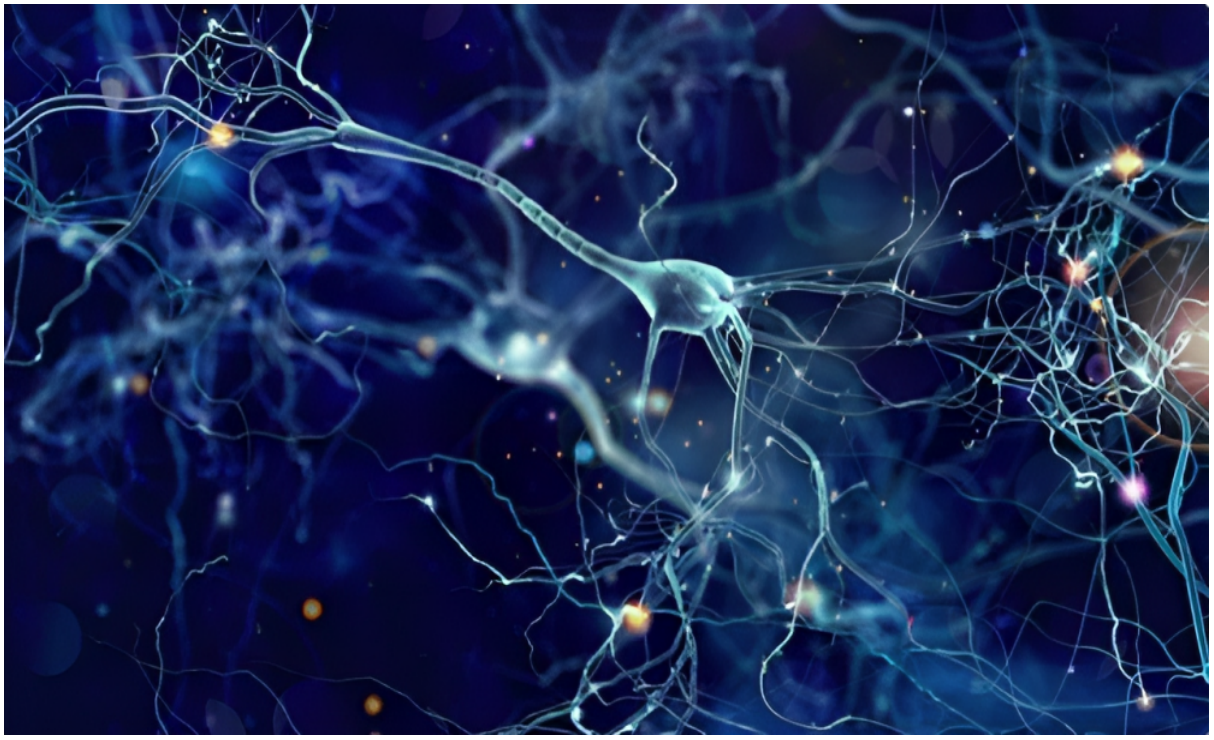
Νευρωνικά Δίκτυα - Βαθιά Μάθηση
Εργασία 2



Ονοματεπώνυμο: Μαχμουτάι Έλενα
Τμήμα: Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
ΑΕΜ: 10012

1. Υλοποίηση της Εργασίας

Σκοπός της εργασίας είναι η υλοποίηση διαφόρων Support Vector Machine Classifications προκειμένου να μελετηθούν οι επιδόσεις τους στον διαχωρισμό 2 κλάσεων που υπάρχουν στην CIFAR-10 βάση. Οι 2 κλάσεις που επιλέχθηκαν από το Dataset προκειμένου να διαχωριστούν ήταν οι *Airplane* και *Automobile*.

Για την υλοποίηση της εργασίας χρησιμοποιήθηκε η γλώσσα Python και οι βιβλιοθήκες keras, numpy, sklearn για την υλοποίηση των SVM και η matplotlib για την απεικόνιση των παραδειγμάτων.

2. Πρώτο Πείραμα

Στο πρώτο πείραμα που βρίσκεται στο αρχείο *linear_svm_from_scratch.py* υλοποιήθηκε απο την αρχή ένας SVM Classifier ο οποίος είχε σκοπό να διαχωρήσει επιτυχώς τα δεδομένα από 2 διαφορετικές κλάσεις του CIFAR-10 Dataset.

Στην αρχή του ταξινομητή φορτώνουμε τα δεδομένα μας χρησιμοποιώντας τη βιβλιοθήκη Keras τα οποία έπειτα ‘ετοιμάζουμε’ για να μπορέσουμε να τα επεξεργαστούμε. Διακρίνουμε τις δύο κλάσεις που θέλουμε από τις υπόλοιπες και κρατάμε μόνο αυτά τα δεδομένα, ενώ έπειτα κάνουμε ανασχηματισμό των δεδομένων (flatten) προκειμένου να αλλάξουμε την διάσταση τους (Από αρχική διάσταση 32X32X3 σε 3072).

Στην συνέχεια ορίζουμε την SVM κλάση μας με τις μεθόδους *__init__ (initialize), fit, update_weights* και *predict*.

- Η μέθοδος *__init__* καλείται κάθε φορά που θέλουμε να αρχικοποιήσουμε μια νέα κλάση και έχει σκοπό την αρχικοποίηση των παραμέτρων *learning_rate, iterations*, *lambda_parameter*.
- Η μέθοδος *fit(self, X, Y)* που έχει σκοπό να εκπαιδεύσει τον SVM Classifier στα δεδομένα που θα του δοθούν αρχικοποιώντας τις παραμέτρους weights και bias και ενημερώνοντας τες κάθε φορά καλώντας την update_*weights* μέσω ενός βρόγχου για συγκεκριμένο αριθμό των επαναλήψεων (iterations) που έχουμε ορίσει.
- Η μέθοδος *update_weights(self)* είναι υπεύθυνη για την ενημέρωση των παραμέτρων *weights* και *bias* με βάση την συνάρτηση απώλειας του SVM. Υπολογίζει τις κλίσεις *dw* και *db* και ενημερώνει τα βάρη χρησιμοποιώντας gradient descent.
- Η μέθοδος *predict(self, X)* προβλέπει τις ετικέτες για τα δεδομένα εισόδου με βάση τα βάρη τους και το bias. Υπολογίζει το output και χρησιμοποιεί την sign συνάρτηση στα προβλεπόμενες ετικέτες για να τις κατατάξει σε αρνητικές και θετικές. Τέλος, όσες ετικέτες είναι μικρότερες ή ίσες με -1 κατατάσσονται στην ετικέτα που έχουμε αντιστοιχίσει με το 0 (Airplane), ενώ οι υπόλοιπες με 1 (Automobile).

Στην συνέχεια του αρχείου καλούμε την κλάση και δημιουργούμε ένα αντικείμενο με τις παραμέτρους που θέλουμε. Ξεκινάμε την καταμέτρηση του χρόνου και κάνουμε fit τον classifier μας. Σταματάμε τον χρόνο και υπολογίζουμε πόσο χρειάστηκε να εκπαιδευτεί το μοντέλο μας. Έπειτα, ξεκινάμε πάλι τον χρόνο και υπολογίζουμε πόσο χρόνο χρειάστηκε ο αλγόριθμος μας να προβλέψει τα δεδομένα train και υπολογίζουμε την ακρίβεια του

αλγορίθμου σε αυτά τα δεδομένα. Με τον ίδιο τρόπο υπολογίζουμε και τον χρόνο που χρειάζεται για να προβλέψει τα testing δεδομένα καθώς και την ακρίβεια του σε αυτά.

Τέλος με την βοήθεια της βιβλιοθήκης *matplotlib* εμφανίζουμε 15 παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης των testing δεδομένων. Τα παραδείγματα αυτά μπορούμε να τα δούμε παρακάτω:

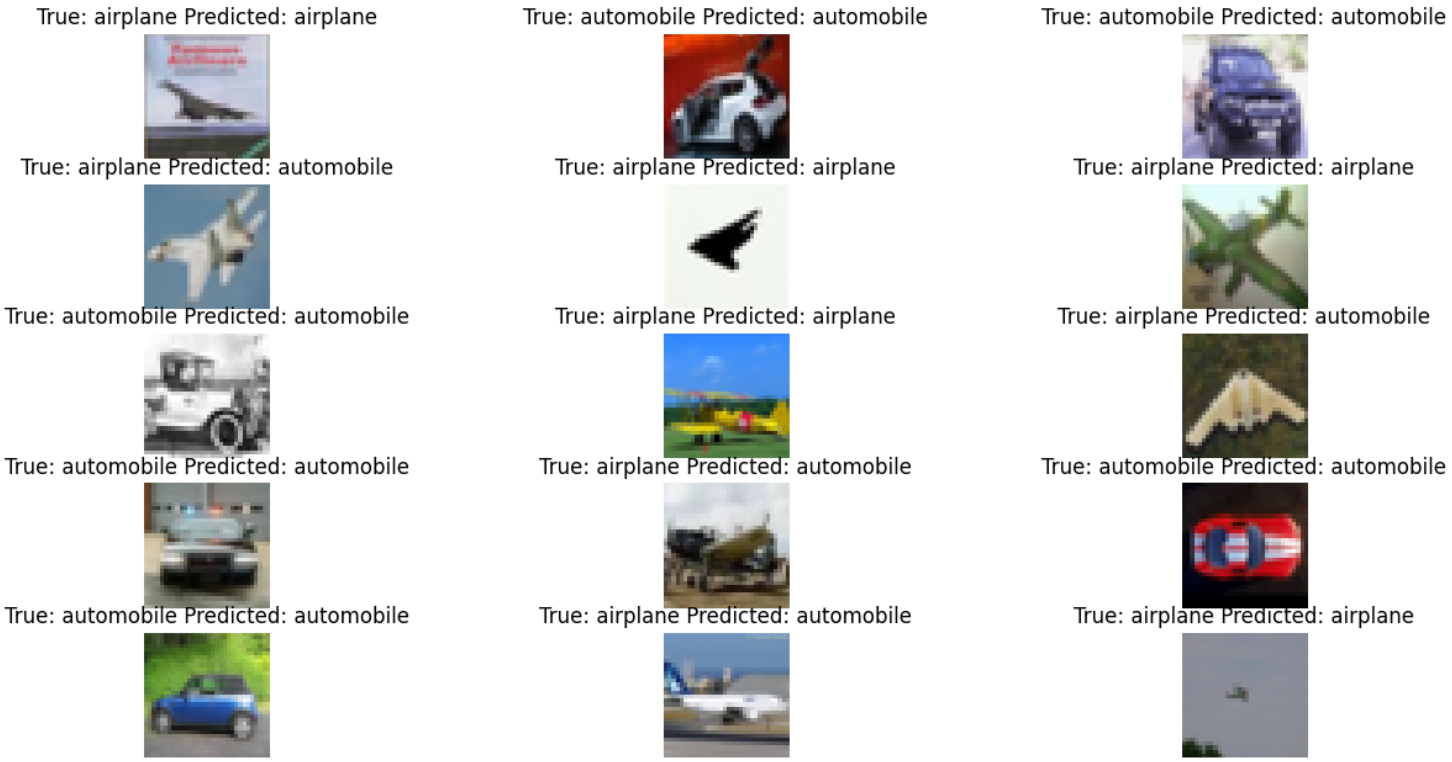


Figure 1: Examples of wrong and right classifications on testing data

Όπως παρατηρούμε ο αλγόριθμος μας κατάφερε με επιτυχία να κατηγοριοποιήσει σωστά τα περισσότερα παραδείγματα ενώ οι φορές που αστόχησε ήταν ελάχιστες. Παρόλα αυτά τα αποτελέσματα στο training και στο testing dataset δεν είναι πολύ υψηλά.

```
Training time: 210.9435911178589 seconds
Prediction time train images: 0.0258786678314209 seconds
the accuracy score on training data is: 0.692
Prediction time test images:: 0.0054094791412353516 seconds
the accuracy score on testing data is: 0.6935
```

Figure 2: Console output of the linear_svm_from_scratch file

3. Δεύτερο Πείραμα

Στο δεύτερο πείραμα το οποίο βρίσκεται στο αρχείο με όνομα *linear_svm.py* υλοποιήθηκε γραμμική ταξινόμηση των κλάσεων με την χρήση των ταξινομητών SVM (SVC και NuSVC) στο υποσύνολο των δεδομένων του CIFAR-10 Dataset. Επιπλέον, εμπεριέχει μετρήσεις απόδοσης και αξιολόγησης ακρίβειας προκειμένου να μπορέσουμε να βγάλουμε συμπεράσματα σχετικά με την αποδοτικότητα του ταξινομητή.

Στην αρχή φορτώνουμε τα δεδομένα μας απο το Dataset και τα προετοιμάζουμε όπως και στο προηγούμενο πείραμα με την διαφορά ότι χρησιμοποιούμε την συνάρτηση ravel για να κάνουμε *flatten* τους πίνακες που περιέχουν τις ετικέτες των *testing* και *training data* σε μονοδιάστατους πίνακες.

Έπειτα ορίζουμε την συνάρτηση *plot_images* για να μπορέσουμε να κάνουμε μια οπτικοποίηση 15 παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης των testing δεδομένων για τον καθένα από τους ταξινομητές μας.

Η συνάρτηση *train_and_evaluate_classifier* έχει σκοπό την εκπαίδευση ενός ταξινομητή. Παρέχει δεδομένα εκπαίδευσης, μέτρησης του χρόνου που απαιτείται για την εκπαίδευση, την πρόβλεψη ετικετών στα σύνολα training και testing, τον υπολογισμό της ακρίβειας και την οπτικοποίηση των αποτελεσμάτων μέσω της συνάρτησης plot_images. Στόχος του είναι να παρέχει μια ολοκληρωμένη επισκόπηση της απόδοσης του ταξινομητή, και να διευκολύνει την αξιολόγηση και τη σύγκριση διαφορετικών ταξινομητών.

Τέλος, αφού έχουμε ορίσει τις παραπάνω συναρτήσεις με την βοήθεια της βιβλιοθήκης from *sklearn* δημιουργούμε ένα αντικείμενο τύπου SVM Classifier χρησιμοποιώντας την συνάρτηση *svm.SCV* και έναν NuSVM Classifier και καλούμε την συνάρτηση *train_and_evaluate_classifier* που έχουμε ορίσει παραπάνω για την ανάλυση των ταξινομητών.

Όπως παρατηρούμε παρακάτω οι ταξινομητές έχουν μεγαλύτερη ακρίβεια απο τον *linear_svm_from_scratch* ταξινομητή που ορίσαμε παραπάνω και έχουν καταφέρει να ταξινομήσουν με επιτυχία σχεδόν όλες τις εικόνες που βρίσκονται στα παραδείγματα. Παρατηρούμε ότι ο *svm.SVC* ταξινομητής έχει μεγαλύτερη ακρίβεια στα training δεδομένα, ενώ ο *NuSVM* ταξινομητής στα testing. Οι χρόνοι που κάνουν για να εκπαιδεύσουν τον ταξινομητή είναι μικρότεροι από το προηγούμενο πείραμα αλλά οι χρόνοι που χρειάζονται για να πραγματοποιήσουν τις προβλέψεις τους είναι αρκετά μεγαλύτεροι.

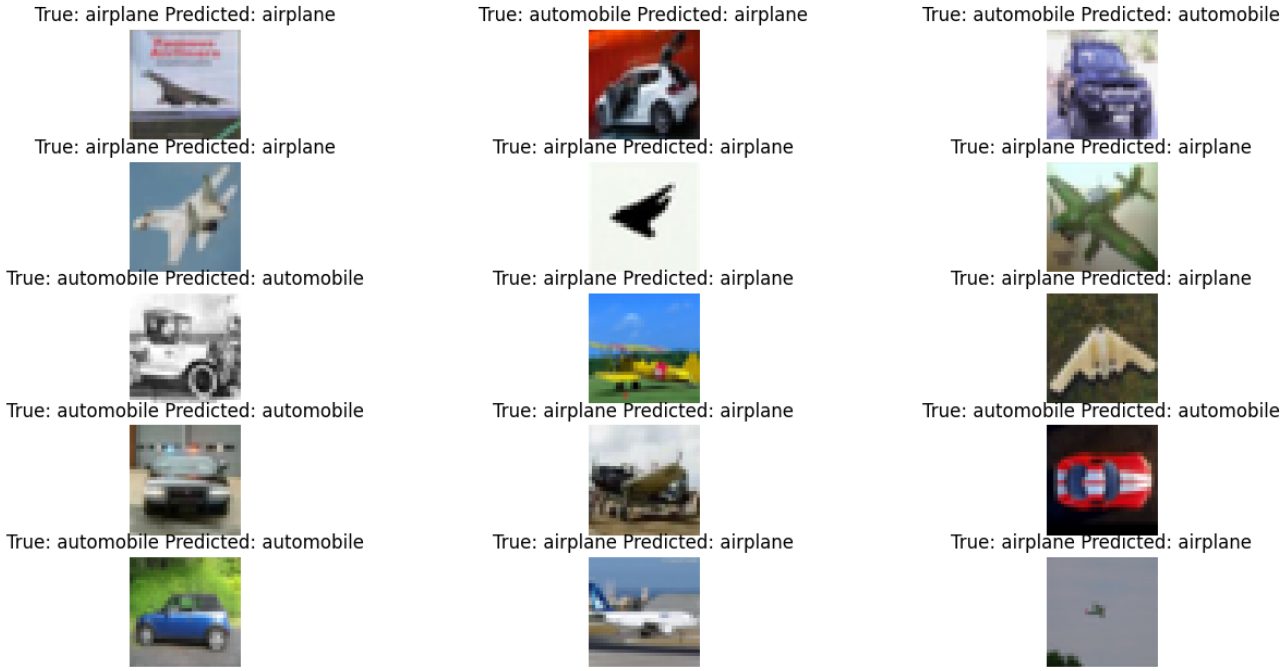


Figure 3: Examples of wrong and right classifications for the SVC on testing data

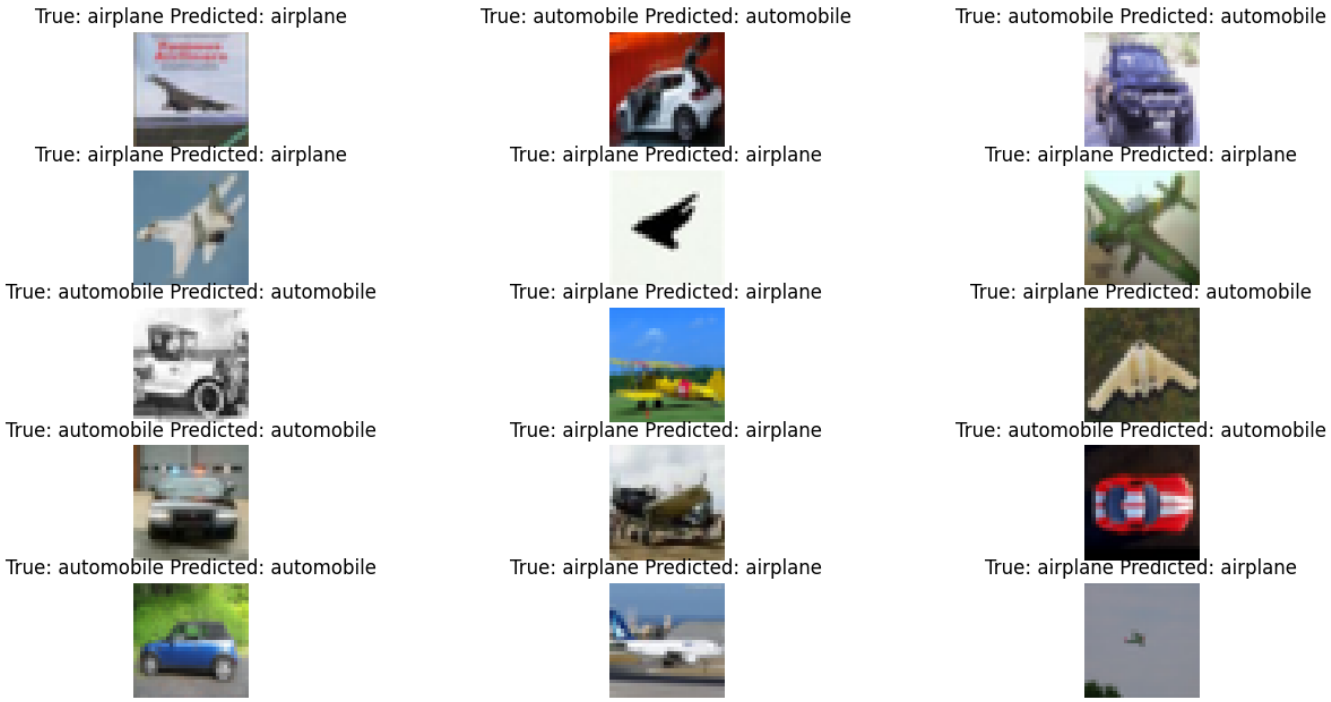


Figure 4: Examples of wrong and right classifications for the NuSVC on testing data

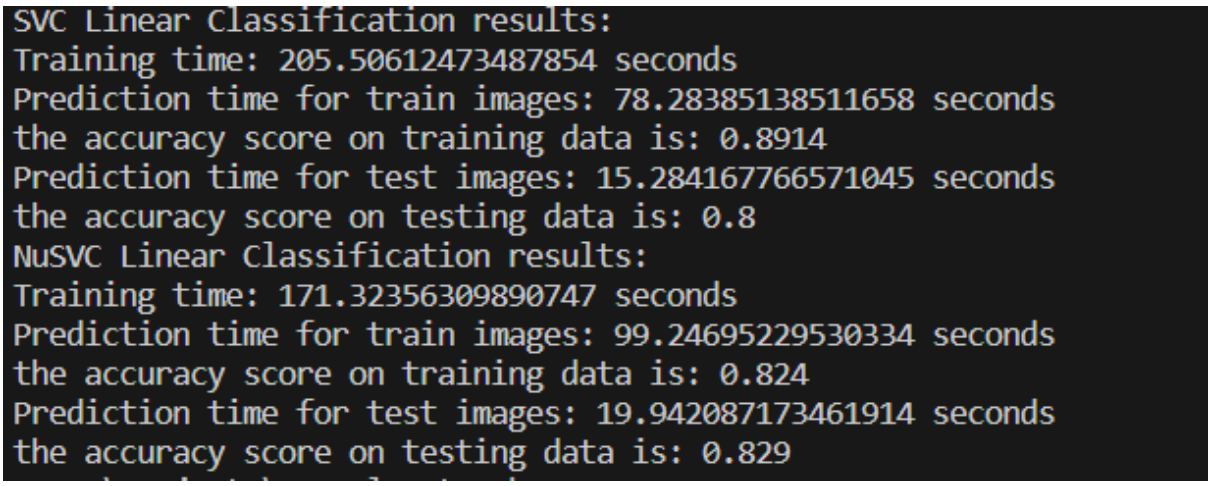


Figure 5: Console output of the linear_svm file

4. Τρίτο Πείραμα

Σκοπός του τρίτου πειράματος που βρίσκεται στο αρχείο με όνομα *rbf_svm.py* είναι να πραγματοποιήσουμε μία ανάλυση του πώς επηρεάζουν διαφορετικοί συνδυασμοί C και gamma έναν ταξινομητή SVM με πυρήνα RBF και να μετρήσουμε τους χρόνους εκπαίδευσης, τους χρόνους πρόβλεψης και την ακρίβεια του ταξινομητή προκειμένου να τα συγκρίνουμε με τα προηγούμενα πειράματα που πραγματοποιήσαμε. Για αυτό τον λόγο αφού ανεβάσουμε τα δεδομένα μας και τα προετοιμάσουμε κατάλληλα όπως και στα προηγούμενα πειράματα, επιλέγουμε διαφορετικές τιμές *C* και *gamma* με εύρος τιμών από 0.001 έως 1000.

Όπως και στο προηγούμενο πείραμα η εκπαίδευση του ταξινομητή και τα αποτελέσματα υπολογίζονται στην συνάρτηση *train_and_evaluate_classifier* η οποία έχει την ίδια λογική όπως και στο προηγούμενο πείραμα και την οποία καλούμε για όλες τις διαφορετικές περιπτώσεις των *gamma* και *C* που έχουμε ορίσει.

Για περισσότερη ευκολία στην ανάλυση αποφασίσαμε να εκτυπώσουμε τα δεδομένα σε ένα νέο αρχείο *txt* το οποίο δημιουργήσαμε στο αρχείο μας. Η απεικόνηση του αρχείου *svm_results.txt* πραγματοποιείται στο αρχείο *rbf_svm_plot.py.*

Η διαδικασία που ακολουθούμε στο αρχείο είναι αρχικά να ξεχωρίσουμε τα δεδομένα μας από κάθε γραμμή του txt file. Αποφασίζουμε ότι η αναπαράσταση που θέλουμε είναι 2D με τον άξονα x να απεικονίζει τις τιμές του C και άξονα y το αποτέλεσμα που θέλουμε να εξετάσουμε κάθε φορα. Το χρώμα της γραφικής παράστασης απεικονίζει την τιμή της μεταβλητής gamma. Οι γραφικές παραστάσεις υλοποιούνται στην συνάρτηση *plot_func* η οποία προσαρμόζεται ανάλογα με το τί θέλουμε να αναπαραστήσουμε κάθε φορά. Παρακάτω βλέπουμε τις γραφικές παραστάσεις που υλοποιούνται στο αρχείο μας.

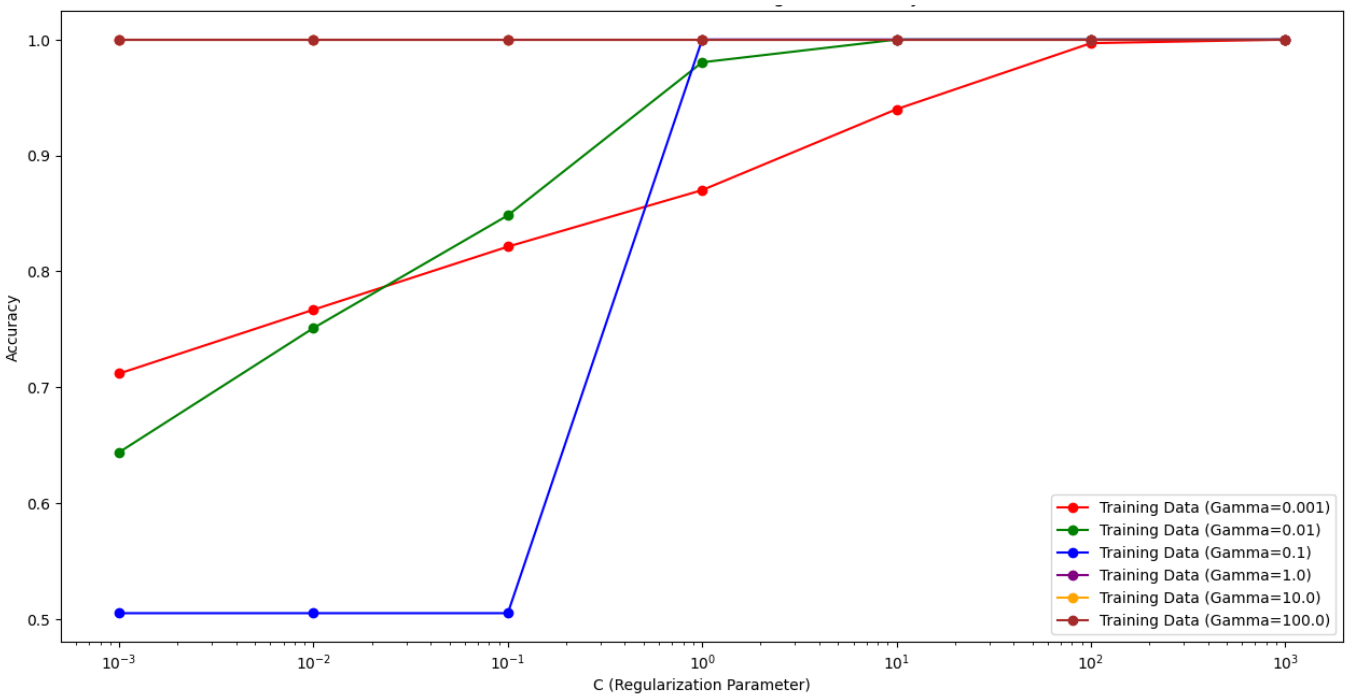


Figure 6: SVM with 'rbf' kernel Classification Training Data Accuracy plot

Στην παραπάνω γραφική παράσταση παρατηρούμε ότι αυξάνοντας την τιμή του *C* αυξάνεται και η ακρίβεια του ταξινομητή για τα training δεδομένα εκτός της περίπτωσης που το *gamma= 100, gamma =10 και gamma = 1* όπου η απόδοση διατηρείται στο μέγιστο ανεξαρτήτως *C*. Επιπλέον, παρατηρούμε ότι στην περίπτωση που το *gamma* έχει τιμή μικρότερη από 1 για μικρες τιμές του *C* ο ταξινομητής έχει μεγαλύτερη απόδοση όσο μικρότερη είναι η τιμή του *gamma* ενώ για μεγαλύτερες συμβαίνει το αντίθετο μέχρι να φτάσει το *C = 100* όπου και όλοι οι ταξινομητές αποκτούν μέγιστη ακρίβεια στο training dataset ανεξαρτήτως *gamma.*

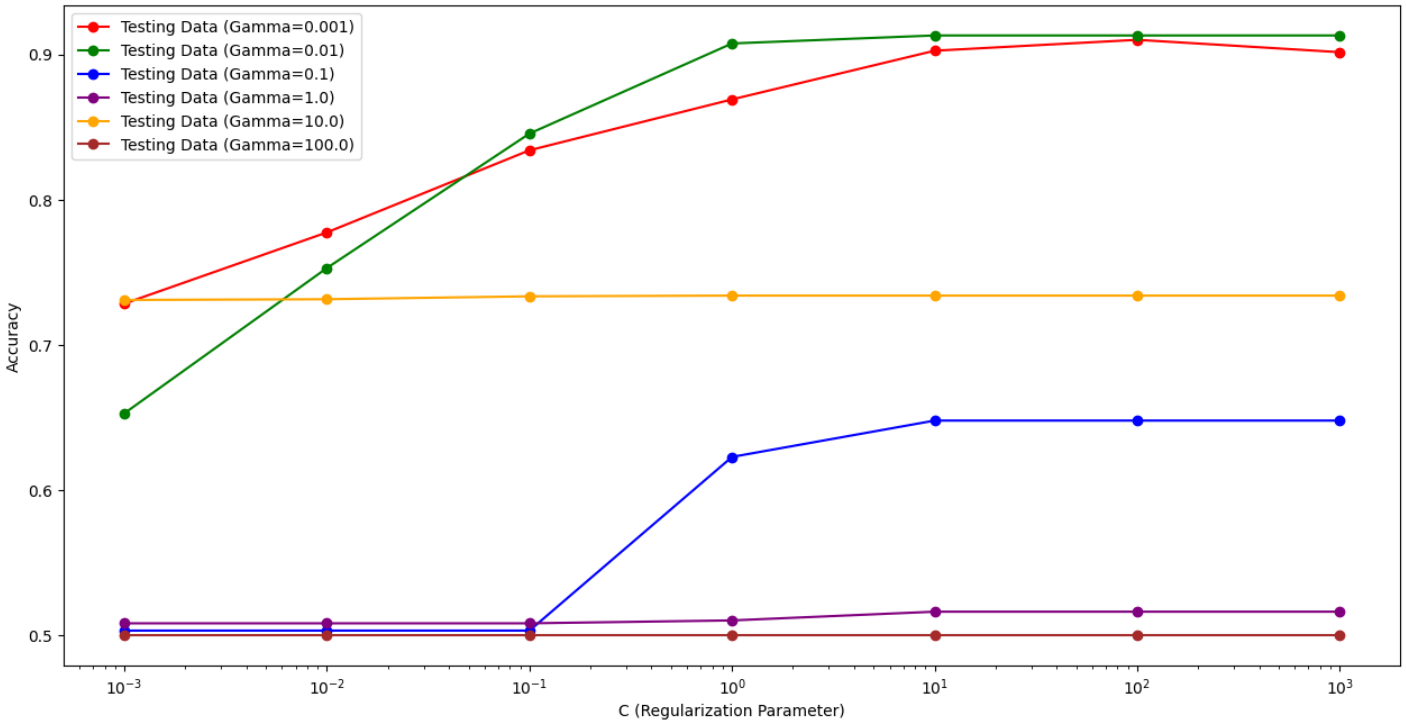


Figure 7: SVM with 'rbf' kernel Classification Testing Data Accuracy plot

Σε αντίθεση με τα training δεδομένα, παρατηρούμε ότι στα testing δεδομένα έχει μεγαλύτερο ρόλο η τιμή του *gamma* παρά η τιμή του *C* στον καθορισμό της ακρίβειας του ταξινομητή. Για τιμές μεγαλύτερες από 1 ο ταξινομητής παρουσιάζει σταθερά αποτελέσματα με μικρές έως μηδενικές μεταβολές για αλλαγή της μεταβλητής *C,* ενώ για μεγαλύτερες τιμές παρατηρούμε πιό ραγδαίες αλλαγές. Συγκριτικά με πριν, όπως θα περιμέναμε, κανένας απο τους ταξινομητές δεν φτάνει στην μέγιστη απόδοση, ενώ σε αυτή την περίπτωση την υψηλότερη απόδοση την έχει η συνάρτηση για *gamma = 0.01* και *C > 1.*

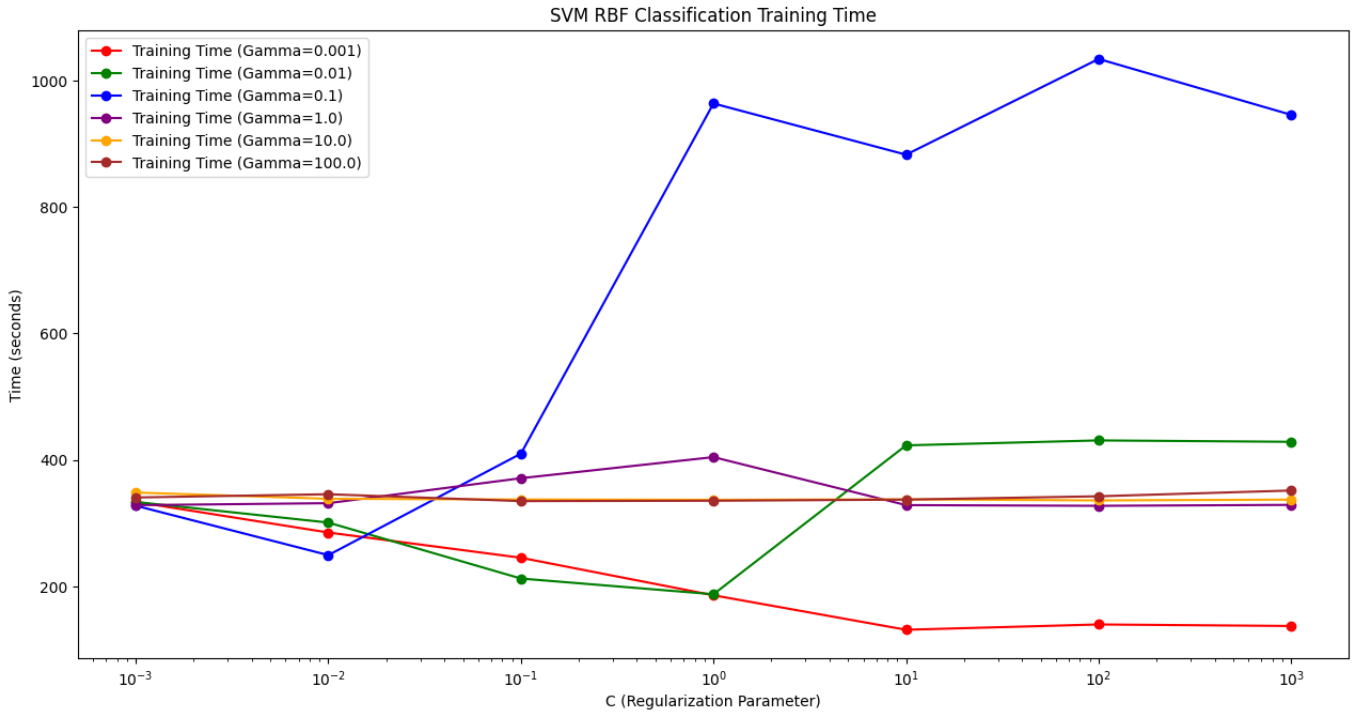


Figure 8: SVM 'rbf' Classification Training time

Όσον αφορά τον χρόνο, παρατηρούμε ότι για *C = 0.001* όλοι οι ταξινομητές χρειάζονται παρόμοιο χρόνο για την εκπαίδευσή τους ενώ εμφανίζουν αρκετές διαφορές όσο η τιμή του *C* μεταβάλλεται. Την μεγαλύτερη αύξηση στον χρόνο εκπαίδευσης εμφανίζει ο ταξινομητής με *gamma = 0.1* ο οποίος τριπλασιάζει τον χρόνο του με την αύξηση του *C*, ενώ για τις τιμές του *gamma = 100* και *1* παρατηρούμε πολύ μικρότερες μεταβολές έως και μηδενικές σε ορισμένες περιπτώσεις. Τον λιγότερο χρόνο εκπαίδευσης παρατηρούμε για τιμές του *gamma = 0.001* και *C=10* όπου το Training Time = 130 seconds.

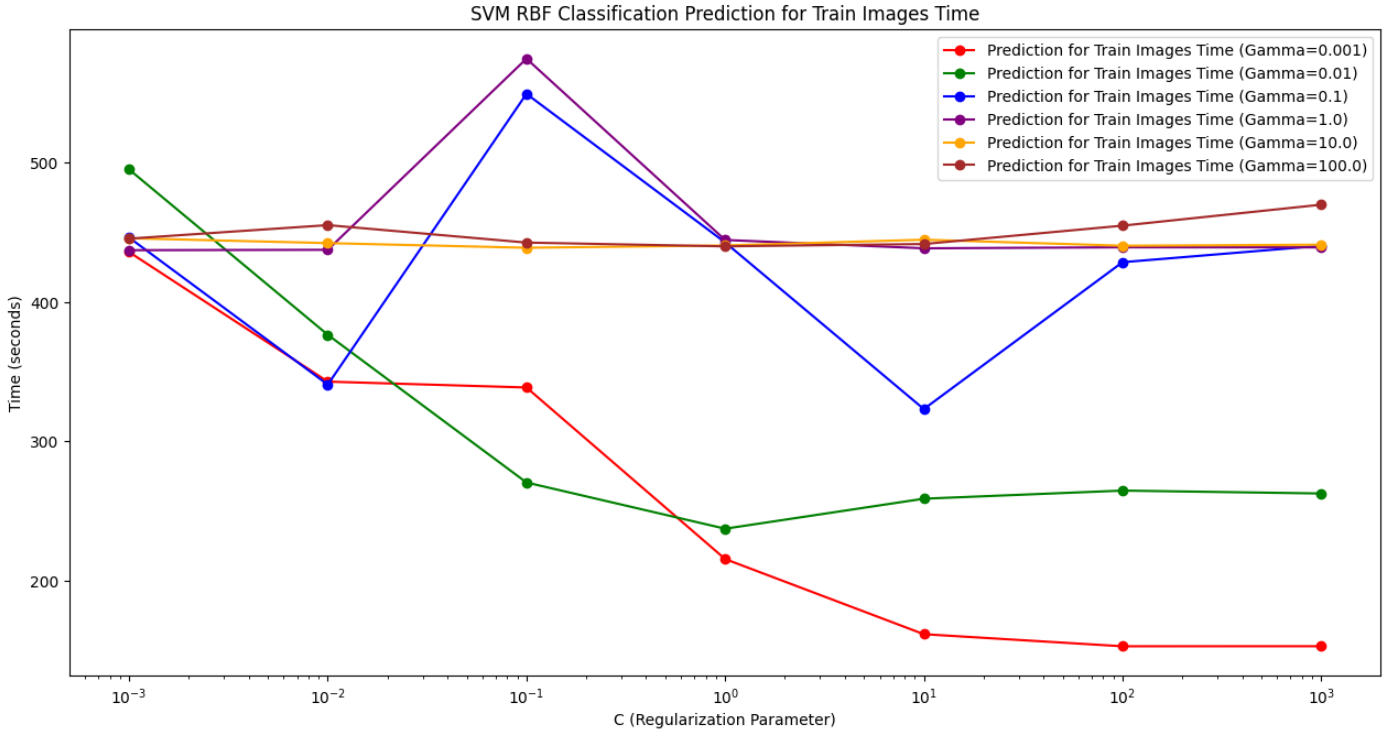


Figure 9: SVM 'rbf' Classification Prediction Time for training images

Όπως και στην προηγούμενη γραφική παρατηρούμε ότι για *gamma = 10 και 100* δεν υπάρχουν σημαντικές μεταβολές στο χρόνο που χρειάζεται για να πραγματοποιήσει πρόβλεψη ο ταξινομητής όσο μεταβάλλεται η τιμή της μεταβλητής *C,* ενώ για τις περιπτώσεις όπου gamma = 0.01 και 0.001 βλέπουμε ραγδαία μείωση του χρόνου με την αύξηση του *C.* Για *gamma = 100* και *gamma = 1* παρατηρούμε έντονες μεταβολες στον χρόνο πρόβλεψης οι οποίες δεν φαίνονται να κοινούνται προς κάποια κατεύθυνση.

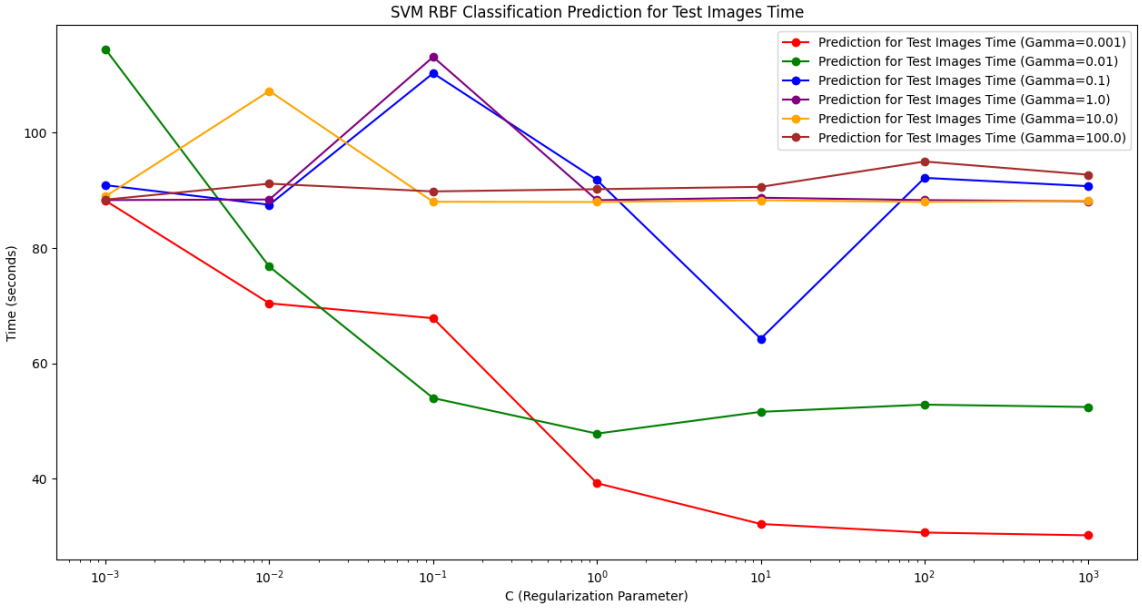


Figure 10: SVM 'rbf' Classification Prediction Time for testing images

Τα αποτελέσματα του χρόνου πρόβλεψης των *testing* δεδομένων ακολουθούν παρόμοια λογική με αυτά των training με τον ταξινομητή που έχει *gamma = 0.001* και *C = 1000* να έπιτυγχάνει τον μικρότερο χρόνο πρόβλεψης.

5. Κατηγοριοποίηση πλησιέστερου γείτονα

Ο κώδικας για την κατηγοριοποίηση πλησιέστερου γείτονα για *k = 1* και *k = 3* για τις κλάσεις Airplane και Automobile υλοποιήθηκε στο αρχείο knn_and_nearest_center_classifiers.py και ακολουθεί την ίδια λογική όπως στην προηγούμενη εργασία.

Τα αποτελέσματα από τον K-Nearest Neighbour για *k = 1* και *k = 3* είναι τα παρακάτω:

```
Accuracy of KNN Classifier with k = 1 : 71.50%
Confusion Matrix for k= 1 :
[[949  51]
 [519 481]]
The classification Report for k = 1 :
              precision    recall  f1-score   support

    airplane       0.65      0.95      0.77      1000
  automobile       0.90      0.48      0.63      1000

    accuracy                           0.71      2000
   macro avg       0.78      0.71      0.70      2000
weighted avg       0.78      0.71      0.70      2000

Accuracy of KNN Classifier with k = 3 : 69.45%
Confusion Matrix for k= 3 :
[[974  26]
 [585 415]]
The classification Report for k = 3 :
              precision    recall  f1-score   support

    airplane       0.62      0.97      0.76      1000
  automobile       0.94      0.41      0.58      1000

    accuracy                           0.69      2000
   macro avg       0.78      0.69      0.67      2000
weighted avg       0.78      0.69      0.67      2000
```

Figure 11: K-Nearest Neighbour Console output

Με βάση τα παραπάνω αποτελέσματα για *k = 1* παρατηρούμε υψηλή ακρίβεια στην κατηγορία "airplane" (95% recall), αλλά χαμηλή στην "automobile" (48% recall). Η συνολική ακρίβεια είναι ικανοποιητική, αλλά η ανισορροπία στα precision και recall είναι κάτι που μας κάνει εντύπωση. Από τα παραπάνω αποτελέσματα μπορούμε να συμπεράνουμε οτι υπάρχει

πιθανότητα ο ταξινομητής να έχει προβλήματα στην αναγνώριση της κατηγορίας "automobile".

Για την περίπτωση *k = 3* παρατηρούμε χαμηλότερη ακρίβεια συγκριτικά με τον KNN Classifier με k=1. Βλέπουμε ότι ο ταξινομητής εξακολουθεί να έχει χαμηλή ακρίβεια στην κλάση "automobile", αν και βελτιώνει την αναγνώριση της σε σχέση με τον *k = 1*.

6. Κατηγοριοποίηση Πλησιέστερου ταξινομητή κέντρου

Ο κώδικας για την κατηγοριοποίηση πλησιέστερου ταξινομητή κέντρου για τις κλάσεις Airplane και Automobile υλοποιήθηκε στο αρχείο knn_and_nearest_center_classifiers.py και ακολουθεί την ίδια λογική όπως στην προηγούμενη εργασία. .

Τα αποτελέσματα από τον Nearest Center Classifier είναι τα παρακάτω:

```
Accuracy of Nearest Center Classifier: 72.30%
Confusion Matrix for Nearest Center Classifier:
[[716 284]
 [270 730]]
The Classification Report for Nearest Center Classifier:
              precision    recall  f1-score   support

    airplane       0.73      0.72      0.72      1000
  automobile       0.72      0.73      0.72      1000

    accuracy                           0.72      2000
   macro avg       0.72      0.72      0.72      2000
weighted avg       0.72      0.72      0.72      2000
```

Figure 12: Nearest Center Classifier *Console output*

Με βάση τα παραπάνω αποτελέσματα παρατηρούμε αρκετά καλή ακρίβεια και ισορροπία μεταξύ των δύο κατηγοριών. Εμφανίζει σχετικά υψηλές τιμές precision, recall και f1-score και για τις δύο κατηγορίες έχοντας μια συνολικά ικανοποιητική απόδοση χωρίς έντονες ανισορροπίες.

7. Συμπεράσματα

Με βάση τα πειράματα που πραγματοποιήθηκαν στο πλαίσιο αυτής της εργασίας, παρατηρούμε πολύ σημαντικά συμπεράσματα σχετικά με την εφαρμογή των ταξινομητών SVM σε προβλήματα κατηγοριοποίησης εικόνων.

Καταρχάς, η χρήση των ενσωματωμένων ταξινομητών SVM, όπως ο svm.SVC και ο NuSVC έχει μεγαλύτερη ακρίβεια σε σχέση με τον ταξινόμητη που υλοποιήσαμε από το μηδεν. Αυτό υπογραμμίζει την αποτελεσματικότητα των ενσωματωμένων εργαλείων στην υλοποίηση τέτοιων αλγορίθμων.

Όσον αφορά τον KNN Classifier, παρατηρούμε ότι ο KNN με k=1 προσφέρει υψηλή ακρίβεια σε ορισμένες κατηγορίες, αλλά εμφανίζει προβλήματα στην αναγνώριση ορισμένων κατηγοριών, ενώ ο Nearest Center Classifier παρουσιάζει ισορροπημένη απόδοση χωρίς έντονες ανισορροπίες. Η επιλογή μεταξύ των δύο εξαρτάται από τις συγκεκριμένες ανάγκες και περιορισμούς του προβλήματος.

Στην περίπτωση του πυρήνα RBF, η επίδραση των παραμέτρων C και gamma είναι καθοριστική. Η αύξηση του C οδηγεί σε βελτίωση της ακρίβειας, ενώ η εύστοχη επιλογή των C και gamma είναι κρίσιμη για την απόδοση στα testing δεδομένα.

Επιπλέον, οι χρόνοι εκπαίδευσης και πρόβλεψης επηρεάζονται σημαντικά από τις τιμές αυτών των παραμέτρων. Η εύρεση της ιδανικής ισορροπίας μεταξύ ακρίβειας και χρόνου είναι ουσιώδης, ιδίως όταν αντιμετωπίζουμε μεγάλα σύνολα δεδομένων.

Συνολικά, η εργασία αυτή επισημαίνει τη σημασία της επιλογής παραμέτρων και του πυρήνα στην απόδοση των ταξινομητών SVM. Η κατανόηση των επιδράσεων αυτών συνεισφέρει στη βελτιστοποίηση της απόδοσης του αλγορίθμου για συγκεκριμένα προβλήματα κατηγοριοποίησης.

Αναφορές

- https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- https://www.pycodemates.com/2022/10/implementing-SVM-from-scratch-in-python.html?utm_content=cmp-true
- https://scikit-learn.org/stable/modules/svm.html
- https://medium.com/pursuitnotes/day-12-kernel-svm-non-linear-svm-5fdefe77836c
- https://matplotlib.org/stable/gallery/subplots_axes_and_figures/gridspec_and_subplots.html#sphx-glr-gallery-subplots-axes-and-figures-gridspec-and-subplots-py
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#sklearn.svm.LinearSVC
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.NuSVC.html#sklearn.svm.NuSVC
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC