Δεύτερη εργασία

Ματάκος Αλέξανδρος

Εισαγωγή

Για την εργασία επιλέχθηκε το πρόβλημα της αναγνώρισης χειρόγραφων ψηφίων. Χρησιμοποιήθηκε η βάση δεδομένων **MNIST** με μορφή που περιγράφεται παρακάτω.
Το πρόγραμμα είναι γραμμένο σε **Python** και υλοποιεί ένα **Support Vector Machine** το οποίο πραγματοποιεί multi-class classification με κλάσεις τα ψηφία 0-9.

Αρχικά έγινε προσπάθεια υλοποίησης from scratch του προβλήματος της αναγνώρισης άρτιων-περιττών ψηφίων, χρησιμοποιώντας μόνο τον **Quadratic solver** της βιβλιοθήκης **cvxopt**. Στην προσπάθεια αυτή υλοποιήθηκε **Linear** καθώς και **Radial Basis Function (RBF)** πυρήνας, δυστυχώς χωρίς επιτυχία, καθώς ό,τι και να δοκίμαζα η μηχανή δεν έδειχνε να μαθαίνει. Μεγάλο μέρος του κώδικα για την προσπάθεια αυτή μπορεί να βρεθεί σε αυτό το link:
https://github.com/cperales/SupportVectorMachine
Ο solver cvxopt χρησιμοποιεί μία interior points μέθοδο η οποία είναι απαιτητική σε υπολογιστικούς πόρους, κυρίως μνήμη. Κατά συνέπεια δεν μπόρεσα να ολοκληρώσω καμία εκπαίδευση με ολόκληρο το dataset, ακόμα και μετά από PCA. Μετά απο πολλές αλλαγές στον κώδικα και αφού πήρα ένα υποσύνολο του dataset πέτυχα σταθερή απόδοση 50-51%, οπότε αποφάσισα να χρησιμοποιήσω την βιβλιοθήκη **scikit-learn**. Ο κώδικας αυτής της προσπάθειας μπορεί να βρεθεί στον φάκελο με όνομα "**scratch**".

Η βάση δεδομένων που χρησιμοποιήθηκε, όπως εξηγείται και στον κώδικα, είναι η **MNIST**, η οποία αποτελείται απο πενήντα χιλιάδες training samples και δέκα χιλιάδες test samples. Τα δείγματα είναι ήδη σε μετασχηματισμένη μορφή (784,1) arrays, με τιμές οι οποίες απο 0-255 είναι scaled down σε 0-1.

Πρόγραμμα

Το πρόγραμμα αποτελείται από δύο αρχεία: Τα ***mnist_loader.py*** και ***test.py.***
Παρακάτω περιγράφεται η λειτουργία του καθενός:

mnist_loader.py:

Το module αυτό είναι υπεύθυνο για τη φόρτωση, την αποθήκευση και την τροποποίηση των δεδομένων που χρησιμοποιεί το πρόγραμμα.
Πιο συγκεκριμένα, η συνάρτηση **load_data** επιστρέφει τα δεδομένα της **MNIST** ως tuple που περιέχει τα training_data και test_data.
Το training_data είναι ένα tuple που αποτελείται από δύο εισόδους. Η πρώτη είσοδος είναι οι 50000 φωτογραφίες των χειρόγραφων ψηφίων σε μορφή (784, 1) arrays, ενώ η δεύτερη είσοδος είναι το ψηφίο που αντιστοιχεί στην κάθε εικόνα.
Τα test_data είναι της ίδιας μορφής, με 10000 (784,1) arrays για το test του νευρωνικού δικτύου.
Η συνάρτηση **load_data_wrapper** φορτώνει αυτά τα δεδομένα. Εδώ ο χρήστης μπορεί να πραγματοποιήσει μείωση της διάστασης των δεδομένων με χρήση PCA (Principal Component Analysis), θέτοντας **use_PCA = True** και εισάγοντας την επιθυμητή διάσταση στη μεταβλητή ***n_components***.
Τέλος, επιστρέφονται ως tuples τα training_data, test_data καθώς και το training dataset χωρίς μειωμένη διάσταση μέσω της μεταβλητής x_test.

test.py:

Το module αυτό εκτελεί το core functionality του προγράμματος.
Αρχικά φορτώνονται τα δεδομένα μέσω της συνάρτησης **load_data_wrapper**, η οποία επιστρέφει τα δεδομένα μειωμένης διάστασης αλλά και αυτά με πλήρη διάσταση. Τα τελευταία χρειάζονται αργότερα ώστε να γίνουν plot κάποια παραδείγματα.
Έπειτα γίνεται η εκπαίδευση του **SVM** με παραμέτρους:

- **C** (penalizing parameter)
- **kernel** (linear, polynomial ή rbf)
- **degree** (βαθμός του πολυονύμου για polynomial η βαθμός προσέγγισης για rbf, δεν παίζει ρόλο στον linear)
- **gamma** (μόνο για τον rbf)

Οι παράμετροι αυτοί καθορίζονται πλήρως από τον χρήστη. Τυπώνεται ο χρόνος εκπαίδευσης σε δευτερόλεπτα.

Στη συνέχεια γίνεται το test, όπου τυπώνονται τα ποσοστά επιτυχίας στο train και στο test. Επιπροσθέτως, ο αλγόριθμος αναγνωρίζει τα **k** το πλήθος παραδείγματα με την χαμηλότερη και υψηλότερη πιθανότητα ορθής κατηγοριοποίησης αντίστοιχα. Τυπώνονται οι εικόνες που αντιστοιχούν στα δείγματα αυτά, καθώς και οι αντίστοιχες πιθανότητες. Η παράμετρος **k** είναι αρχικοποιημένη με τιμή 3. Εδώ βλέπουμε ότι υπάρχουν πολύ "δύσκολα" παραδείγματα, όπου ο αλγόριθμος είναι μόλις κατά 26% confident για την απόφαση του, αλλά και πολύ "εύκολα" παραδείγματα, όπου το confidence φτάνει το 99.9%.

Πειράματα

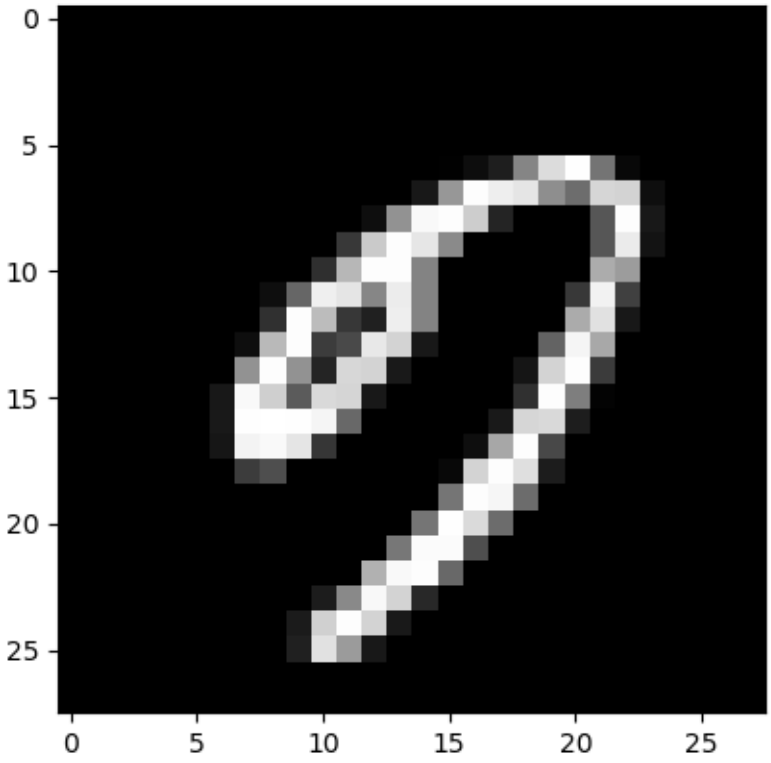
Στα πειράματα χρησιμοποιήθηκαν linear, polynomial και rbf kernels και διάφορες τιμές για τις παραμέτρους. Τα ποσοστά σε όλες τις περιπτώσεις ήταν πολύ ψηλά. Η λειτουργία υψηλότερης/χαμηλότερης πιθανότητας που περιγράφεται στην προηγούμενη παράγραφο παρουσιάζεται μόνο στο πρώτο πείραμα καθώς είναι αρκετά απαιτητική σε υπολογιστικούς πόρους, για λόγους εξοικονόμησης χρόνου.
Δυστυχώς για υψηλές τιμές της παραμέτρου **C** η διαδικασία εκπαίδευσης δεν τελείωνε στη μηχανή μου οπότε δεν έχω αποτελέσματα να παρουσιάσω για αυτή την περίπτωση.

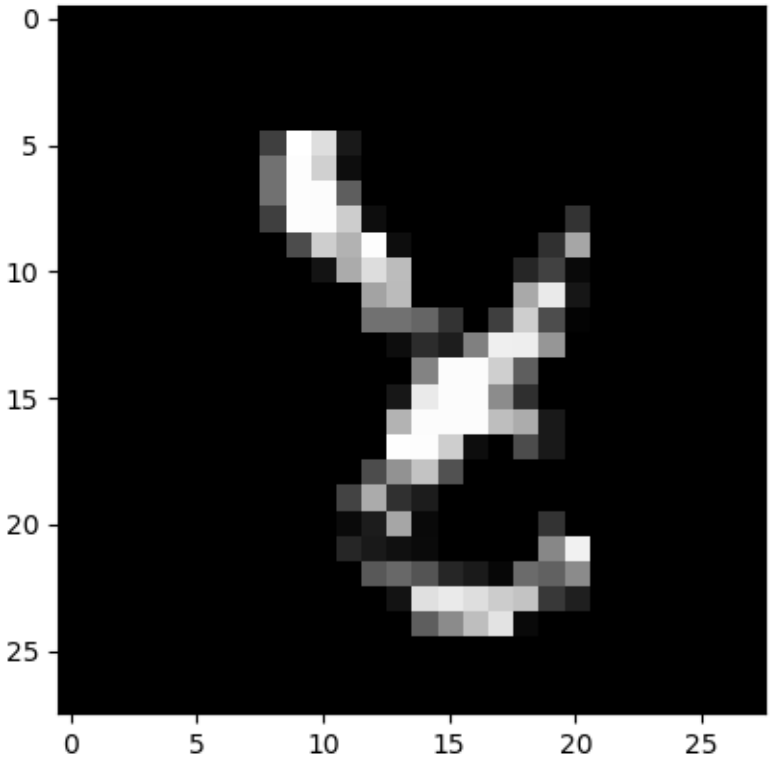
Πείραμα 1

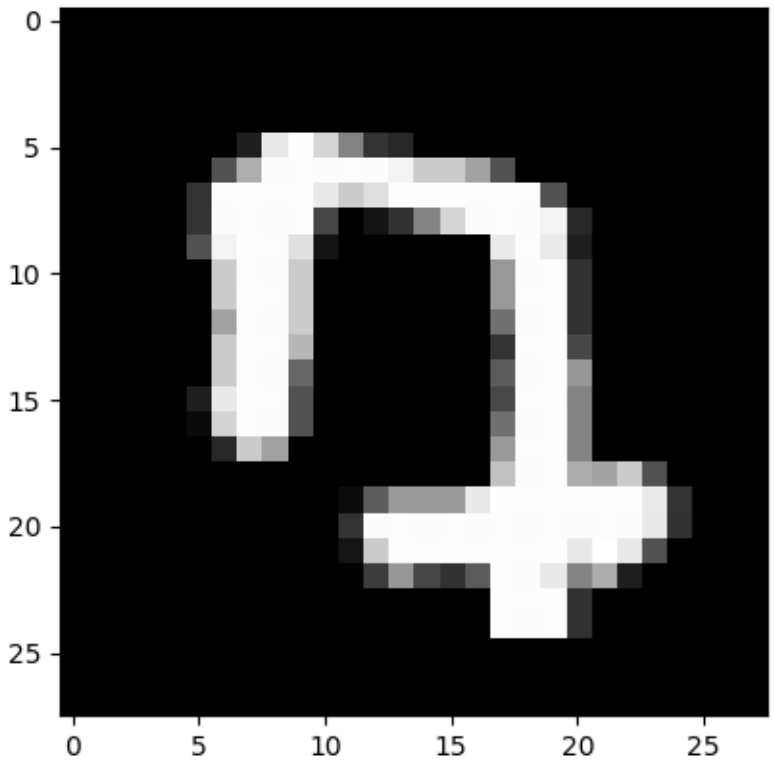
C=2, kernel=rbf, degree=3, gamma=0.023

```
[5 0 4 1 9 2 1 3 1 4]
Using the whole dataset with reduced dimensionality n=50
(50000, 50) (50000,) (10000, 50) (10000,)
Elapsed time for training: 155.39860486984253
Parameters:
C = 2.0
degree = 3
gamma = 0.023
kernel = rbf
Probability of 3most difficult examples:
[0.23908244 0.23652314 0.27394695]
Plotting the images of the most difficult examples
Probability of 3easiest examples:
[0.99966485 0.99941782 0.99891053]
Plotting the images of the easiest examples
Accuracy on training: 0.9845
Accuracy on test: 0.99542
```

Παρακάτω οι εικόνες των τριών παραδειγμάτων με τις χαμηλότερες πιθανότητες ορθής κατηγοριοποίησης:

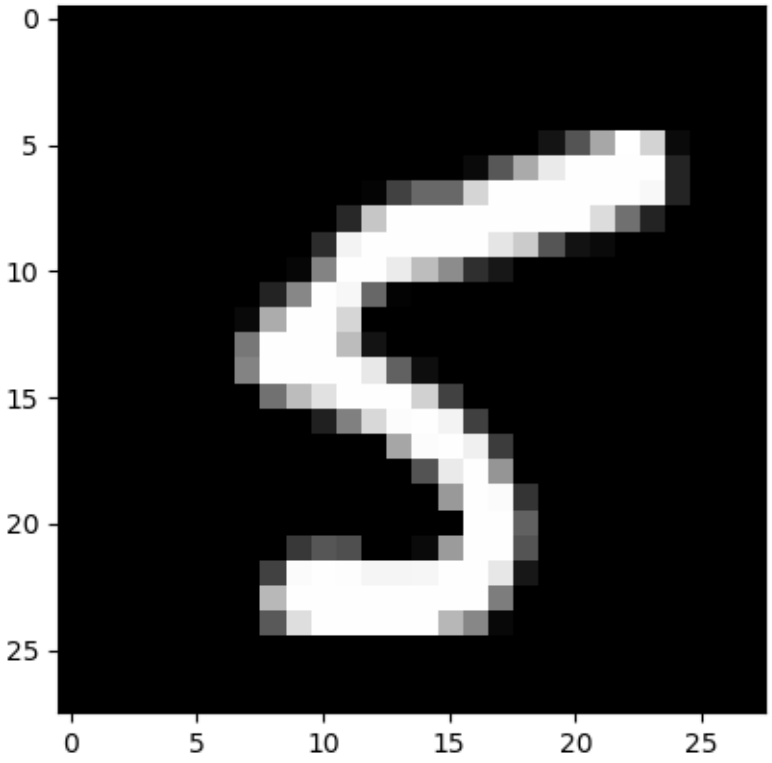


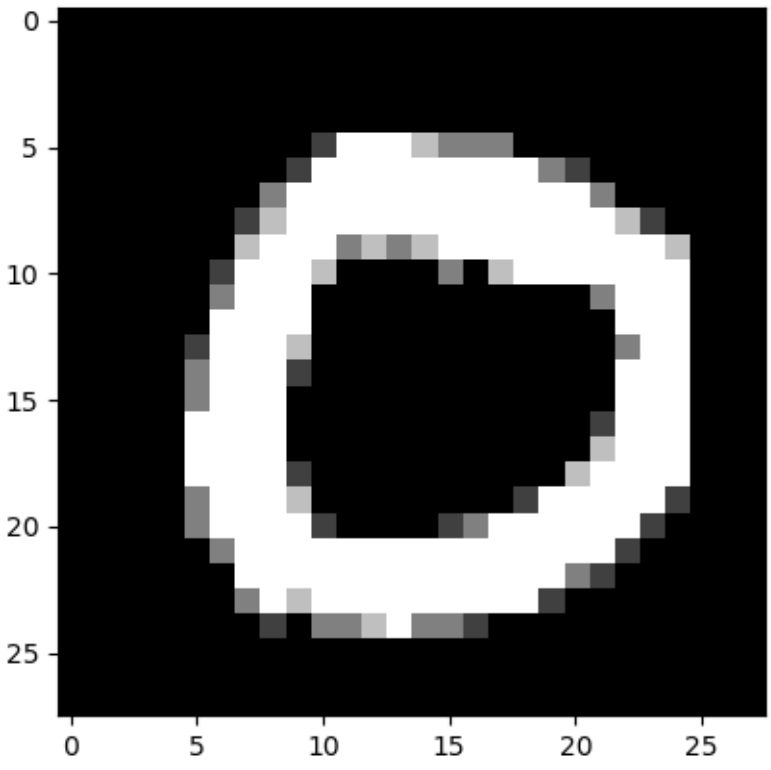

0
5
10
15
20
25
0
5
10
15
20
25

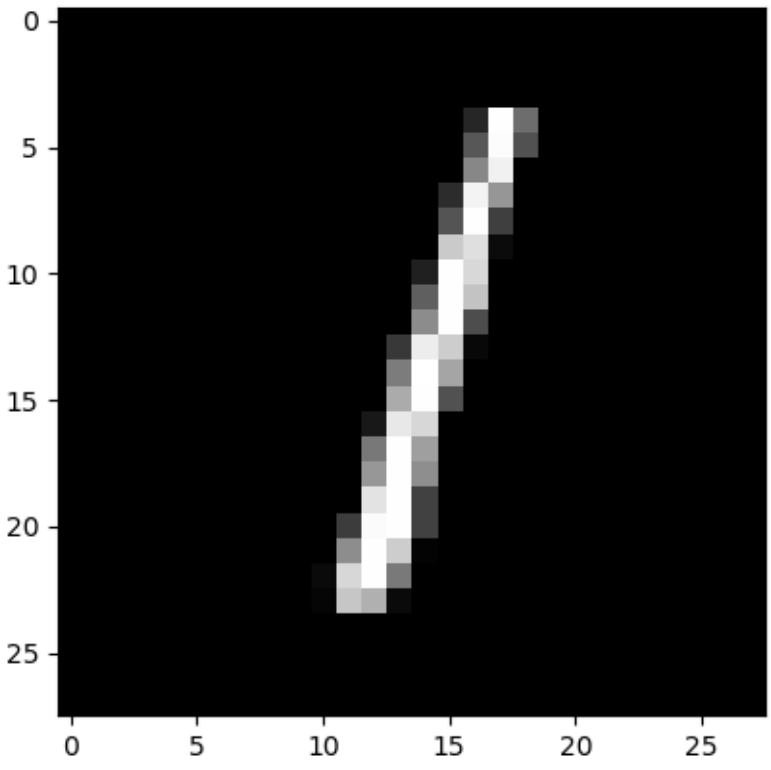


Βλέπουμε και μόνοι μας ότι τα συγκεκριμένα παραδείγματα ήταν πολύ δύσκολα.

Έπειτα οι εικόνες των τριών παραδειγμάτων με τις υψηλότερες πιθανότητες ορθής κατηγοριοποίησης:




0
5
10
15
20
25
0
5
10
15
20
25



Οι αντίστοιχες πιθανότητες περιγράφονται στο πρώτο screenshot.

Πείραμα 2

C=1, kernel=rbf, degree=3, gamma=0.023

```
Using the whole dataset with reduced dimensionality n=50
(50000, 50) (50000,) (10000, 50) (10000,)
Elapsed time for training: 163.53840565681458
Parameters:
C = 1.0
degree = 3
gamma = 0.023
kernel = rbf
Accuracy on training: 0.9822
Accuracy on test: 0.99092
```

Πείραμα 3

C=5, kernel=rbf, degree=3, gamma=0.023

```
Using the whole dataset with reduced dimensionality n=50
(50000, 50) (50000,) (10000, 50) (10000,)
Elapsed time for training: 28.033954858779907
Parameters:
C = 5.0
degree = 3
gamma = 0.023
kernel = rbf
Accuracy on training: 0.9853
Accuracy on test: 0.99872
```

Πείραμα 4

C=2, kernel=rbf, degree=3, gamma=0.1

```
Using the whole dataset with reduced dimensionality n=50
(50000, 50) (50000,) (10000, 50) (10000,)
Elapsed time for training: 195.44925165176392
Parameters:
C = 2.0
degree = 3
gamma = 0.1
kernel = rbf
Accuracy on training: 0.9813
Accuracy on test: 0.99986
```

Δυστυχώς για **gamma** ≥ 1, η διαδικασία εκπαίδευσης δεν ολοκληρώνονταν οπότε το περιθώριο πειραματισμού ήταν μικρό.

Έπειτα χρησιμοποιώντας linear kernel βλέπουμε ότι έχουμε χαμηλότερα ποσοστά σε train και test, κοντά στο 93%.

Πείραμα 5

C=1, kernel=linear

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 48.78967642784119
Parameters:
C = 1.0
degree = 3
gamma = auto_deprecated
kernel = linear
Accuracy on training: 0.9369
Accuracy on test: 0.93664
```

Πείραμα 6

C=5, kernel=linear

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 116.98967576026917
Parameters:
C = 5.0
degree = 3
gamma = auto_deprecated
kernel = linear
Accuracy on training: 0.9363
Accuracy on test: 0.93686
```

Πείραμα 7

C=20, kernel=linear

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 300.6703016757965
Parameters:
C = 20.0
degree = 3
gamma = auto_deprecated
kernel = linear
Accuracy on training: 0.937
Accuracy on test: 0.93662
```

Τέλος, χρησιμοποιούμε κάποιους πολυονυμικούς πυρήνες, οι οποίοι πιάνουν ποσοστά 100% στο test για βαθμό μεγαλύτερο του 2.

Πείραμα 8

C=2, kernel=polynomial, degree=2

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 21.519318103790283
Parameters:
C = 2.0
degree = 2
gamma = 0.1
kernel = poly
Accuracy on training: 0.9794
Accuracy on test: 0.9998
```

Πείραμα 9

C=2, kernel=polynomial, degree=5

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 62.26886463165283
Parameters:
C = 2.0
degree = 5
gamma = 0.1
kernel = poly
Accuracy on training: 0.9773
Accuracy on test: 1.0
```

Πείραμα 10

C=100, kernel=polynomial, degree=3

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 30.97104501724243
Parameters:
C = 100.0
degree = 3
gamma = 0.1
kernel = poly
Accuracy on training: 0.9811
Accuracy on test: 1.0
```

Πείραμα 11

C=20, kernel=polynomial, degree=5

```
Using the whole dataset with reduced dimensionality n=50
Train dataset shapes: (50000, 50), (50000,)
Test dataset shapes: (10000, 50), (10000,)
Elapsed time for training: 63.076897621154785
Parameters:
C = 20.0
degree = 5
gamma = 0.1
kernel = poly
Accuracy on training: 0.9772
Accuracy on test: 1.0
```

Πορίσματα

Βλέπουμε ότι το **SVM** τα έχει πάει εξαίσια, με ποσοστά 93% για **linear kernel**, 99.5% για **rbf kernel** και μέχρι και 100% για **polynomial kernel**!. Συγκρίνοντας με τα αποτελέσματα της Ενδιάμεσης Εργασίας, παρατηρούμε ότι η βέλτιστη απόδοση του **SVM** στο 100% είναι μεγαλύτερη από τις αποδόσεις των **KNeighborClassifier** για αριθμό κοντινότερων γειτόνων **NN=1** και **NN=3**, οι οποίες κυμαίνονταν στο 97%. Για τον **rbf kernel** είναι πάλι υψηλότερα με ποσοστά 99%, ενώ ο **linear kernel** δεν κατάφερε να τον ξεπεράσει καθώς είχε ποσοστά κοντά στο 93%.
Όσο για τον **NearestCentroid** classifier, βλέπουμε ότι το **SVM** τα έχει πάει πολύ καλύτερα από τον **NearestCentroid** σε όλες τις περιπτώσεις, με ποσοστά μεγαλύτερα του **NearestCentroid** που ήταν γύρω στο 82%.