Neural Networks
&
Deep Learning

2Η ΥΠΟΧΡΕΩΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Μυλωνάς | 10027 | 23.12.2022

Word Count: 2504

Πίνακας περιεχομένων

Εισαγωγή

Στην συγκεκριμένη εργασία ζητήθηκαν η υλοποίηση ενός προγράμματος σε οποιαδήποτε γλώσσα προγραμματισμού, το οποίο να συγκρίνει την απόδοση του κατηγοριοποιητή πλησιέστερου γείτονα με 1 και 3 γείτονες (K-Nearest Neighbor Classifier ή KNN) με αυτή του κατηγοριοποιητή πλησιέστερου κέντρου (Nearest Class Centroid Classifier ή NCC) και η υλοποίηση ενός Support Vector Machine (SVM) και η σύγκριση της απόδοσης αυτού με τους παραπάνω κατηγοριοποιητές. Το SVM εκπαιδεύεται για να διαχωρίζει χειρόγραφα ψηφία. Το πρόβλημα αυτό εμπίπτει στη κατηγορία της επίλυσης προβλήματος κατηγοριοποίησης πολλών κλάσεων.

Ο κατηγοριοποιητής KNN υπολογίζει τα πλησιέστερα K σημεία και κατηγοριοποιεί το στοιχείο που μελετάμε με βάση τη συχνότητα εμφάνισης των κατηγοριών αυτών των K σημείων. Είναι σαφές πως για να μην υπάρχει ισοψηφία, επιλέγουμε την παράμετρο K να είναι περιττός αριθμός.

Ο κατηγοριοποιητής NCC αρχικά διαχωρίζει σε κλάσεις τα πιθανά αποτελέσματα (10 στη συγκεκριμένη περίπτωση). Στην συνέχεια υπολογίζει το κέντρο (centroid) της κάθε κλάσης με τη χρήση του μέσου όρου. Τελικά, συγκρίνει την απόσταση του σημείου που μελετάμε από αυτά τα κέντρα, και το κατηγοριοποιεί το σημείο στην κλάση που η απόσταση αυτή είναι η ελάχιστη.

Το SVM είναι ένα μοντέλο εποπτευόμενης μάθησης (supervised learning). Για την κατηγιοριοποίηση σε κλάσεις, όταν ο γραμμικός διαχωρισμός (με τη χρήση LinearSVM) δεν είναι εφικτός, τα δεδομένα ανάγονται σε υψηλότερες διαστάσεις, σε ένα πολυδιάστατο επίπεδο. Η αναγωγή αυτή γίνεται μέσω των συναρτήσεων πυρήνα (kernels). Στην συνέχεια επιλέγει ένα υπερεπίπεδο (hyperplane) ως το καλύτερο δυνατό για τη κατηγοριοποίηση. Το καλύτερο υπερεπίπεδο θεωρείται αυτό που μεγιστοποιεί το περιθώριο (margin) από κάθε κλάση, δηλαδή απέχει το μέγιστο από κάθε κοντινότερο δείγμα.

Η επιλογή του υπερεπιπέδου εξαρτάται εν μέρει από τις παραμέτρους που δίνουμε εμείς στο SVM, όπως η συνάρτηση kernel. Σε γραμμικά προβλήματα (συνήθως 2 κλάσεων) χρησιμοποιείται κυρίως το linear kernel. Σε μη γραμμικά προβλήματα μπορούμε να επιλέξουμε μεταξύ κάποιων άλλων συναρτήσεων όπως η polynomial (poly), η radial basis function (RBF) και άλλες.

Ακόμη, όταν το dataset είναι πολύ μεγάλο, χρησιμοποιούμε Principal Component Analysis (PCA) ώστε να κρατήσουμε όσο το δυνατόν περισσότερη πληροφορία θέλουμε, μειώνοντας ταυτόχρονα τις διαστάσεις του προβλήματος. Με αυτή τη τεχνική, μειώνονται τα components που κρατάμε την πληροφορία και έτσι το πρόβλημα παίρνει μικρότερες διαστάσεις.

Τεχνικές Προδιαγραφές

Για την υλοποίηση χρησιμοποιήθηκε Python ενώ ως περιβάλλον εργασίας το PyCharm.

Ως βάση δεδομένων επιλέχθηκε η MNIST[1] dataset, η οποία περιλαμβάνεται στο Keras[2]. Η MNIST αποτελεί μια συλλογή από χειρόγραφα ψηφία, τα οποία χρησιμοποιούνται για την εκπαίδευση αλγορίθμων επεξεργασίας εικόνας. Το Keras είναι ένα API βαθιάς μάθησης γραμμένο σε Python, που τρέχει πάνω στην πλατφόρμα μηχανικής μάθησης TensorFlow.

Για τους κατηγοριοποιητές (KNN, NCC, SVC) χρησιμοποιήθηκε η βιβλιοθήκη μηχανικής μάθησης scikit-learn ή sklearn[3].

Σαφώς χρησιμοποιήθηκαν και Python modules όπως NumPy και Time.

Η προδιαγραφές της συσκευής που χρησιμοποιήθηκε για την εκτέλεση του κώδικα και τη λήψη μετρήσεων είναι οι εξής:

- Processor: 12th Gen Intel(R) Core(TM) i7-1260P @ 2.10 GHz
- Ram memory: 32 GB
- System type: 64-bit operating system, x64-based processor

[1] MNIST Dataset: http://yann.lecun.com/exdb/mnist/
[2] Keras: https://keras.io/about/
[3] Sklearn: https://scikit-learn.org/stable/index.html

Υλοποίηση

KNN, NCC

Αρχικά εισάγουμε τις απαραίτητες βιβλιοθήκες, ώστε να μπορούμε να χρησιμοποιήσουμε τη βάση δεδομένων, διάφορες μεθόδους και τους κατηγοριοποιητές. Εναλλάσσοντας παραμέτρους στον αλγόριθμο, όπως τον τύπο της απόστασης σημείων ή τον αριθμό των γειτόνων στον KNN, παίρνουμε μετρήσεις σχετικά με το ποσοστό επιτυχίας και το χρόνο εκπαίδευσης και πρόβλεψης. Στη συγκεκριμένη εργασία λήφθηκαν μετρήσεις από 1 έως 25 γείτονες στον KNN και Ευκλείδεια και Manhattan απόσταση στον NCC (για σκοπούς απλής παρατήρησης). Τελικά, συγκρίνουμε τόσο τον χρόνο όσο και την ακρίβεια κάθε αλγορίθμου και καταλήγουμε σε συμπεράσματα.

KNN CROSS VALIDATION

Η βάση δεδομένων περιλαμβάνει 2 σετ, 1 για εκπαίδευση (train set) και 1 για testing (test set). Το train set περιλαμβάνει 50,000 στοιχεία, ενώ το test set περιλαμβάνει 10,000 στοιχεία. Μελετώντας τον KNN περαιτέρω, βλέπουμε πως μπορεί να εμφανιστεί το φαινόμενο overfitting, δηλαδή ο αλγόριθμος να μαθαίνει πολύ εύκολα το dataset που του δίνουμε και να προσαρμόζεται πλήρως σε αυτό, χωρίς να μπορεί να διαχειριστεί διαφορετικά datasets. Επομένως, χρησιμοποιήθηκε και η τεχνική k-fold Cross Validation (CV) για πιο ακριβή αποτελέσματα. Στην CV, ενώνουμε τα train & test sets σε ένα set 60,000 εικόνων. Στη συνέχεια χωρίζουμε σε k folds το συνολικό dataset και χρησιμοποιούμε 1 fold ως test set και τα υπόλοιπα k-1 folds ως train sets. Με αυτόν τον τρόπο εξασφαλίζουμε πως κάθε στοιχείο του συνολικού set των 60,000 στοιχείων θα έχει χρησιμοποιηθεί και για εκπαίδευση και για testing και συνεπώς πως ο αλγόριθμός μας μπορεί να διαχειρίζεται και διαφορετικά datasets.

Στη συγκεκριμένη εργασία χρησιμοποιούμε τον KNN με 3 γείτονες για να πάρουμε μετρήσεις. Το πρόγραμμα που υλοποιείται παίρνει μετρήσεις για 2-39 folds.

ΆΛΛΕΣ ΤΕΧΝΙΚΕΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΤΗΣ ΑΚΡΙΒΕΙΑΣ

Ύστερα από έρευνα γύρω από την απόδοση του KNN, βλέπουμε πως υπάρχουν και άλλες τεχνικές αύξησης της ακρίβειας του κατηγοριοποιητή. Εκτός από την Cross Validation, που αφορά στην υλοποίηση της κατηγοριοποίησης, μπορούμε να επέμβουμε και στο ίδιο το dataset και να το επεξεργαστούμε πριν το χρησιμοποιήσουμε στον αλγόριθμό μας. Μπορούμε να απομακρύνουμε το θόρυβο χρησιμοποιώντας denoising autoencoders, ή να κάνουμε blur (defocus, linear horizontal motion blur). Ωστόσο αυτές οι τεχνικές απλώς ερευνήθηκαν, χωρίς να υλοποιηθούν στη συγκεκριμένη εργασία.

SVM

Για την υλοποίηση του SVM χρησιμοποιούμε τη βιβλιοθήκη sklearn. Συγκεκριμένα, χρησιμοποιήσαμε αρχικά το μοντέλο SVC που είναι ένας classifier, καθώς ο διαχωρισμός των ψηφίων σε κλάσεις είναι πρόβλημα classification. Δοκιμάσαμε και τους linearSVC, που πρακτικά πρόκειται για τον SVC με kernel = 'linear', απλώς εντάσσεται σε άλλη βιβλιοθήκη και παρέχει περισσότερες συναρτήσεις loss και penalties ώστε να κάνει καλύτερο scaling με μεγάλο αριθμό δειγμάτων. Η διαφορά μεταξύ SVC & NuSVC είναι πως το πρώτο μοντέλο χρησιμοποιεί τη παράμετρο C ενώ το δεύτερο την Nu.

Αρχικά το μοντέλο δοκιμάστηκε με τη βιβλιοθήκη με ψηφία της sklearn, η οποία περιλαμβάνει περίπου 2000 δείγματα. Ο χρόνος ήταν αρκετά μικρός ώστε το μοντέλο να κάνει fit. Ωστόσο, όταν δοκιμάσαμε με τη βιβλιοθήκη mnist, η οποία περιλαμβάνει 60.000 δείγματα, δεν πήραμε αποτελέσματα. Για το λόγο αυτό δοκιμάσαμε να εφαρμόσουμε Principal Component Analysis (PCA). Με τον τρόπο αυτό απορρίπτουμε στρατηγικά πολλά components του dataset, ώστε ο χρόνος που απαιτείται ώστε να κάνει fit το μοντέλο να κυμαίνεται σε λογικά πλαίσια. Προκειμένου να μη χάσουμε σε ακρίβεια όμως, μπορούμε να κανονικοποιήσουμε το dataset (standardize the dataset) χρησιμοποιώντας τον Standard Scaler. Ο RBF Kernel στα SVM υποθέτει ότι το dataset έχει κανονικοποιηθεί γύρω από το μηδέν(0) με κάποια διακύμανση. Στη συγκεκριμένη περίπτωση, μειώσαμε μέσω PCA την διαθέσιμη πληροφορία στο 90% μειώνοντας τα components σε 87. Ο μέσος χρόνος μειώθηκε αμέσως από 16 λεπτά σε περίπου 4.

Στη συνέχεια εκτελούμε hyperparameter tuning (συντονισμό υπερπαραμέτρων) εκτελώντας μια Cross Validation. Συγκεκριμένα χρησιμοποιούμε την GridSearchCV μέθοδο της βιβλιοθήκης SKLearn. Εισάγουμε διάφορες τιμές των μεταβλητών C και gamma καθώς και της συνάρτησης kernel. Επιλέξαμε αυτή τη τεχνική για hyperparameter tuning κάνοντας brute force κάθε συνδυασμού παραμέτρων. Πρόκειται για μια exhaustive brute force τεχνική που εξαντλεί κάθε δυνατό συνδυασμό. Η διαδικασία αυτή είναι πολύ χρονοβόρα. Στη συγκεκριμένη περίπτωση χρειάστηκε περίπου 17 ώρες για να ολοκληρωθεί, με k=5 folds για κάθε υποψήφιο. Συνολικά δώσαμε 4 τιμές για τη παράμετρο C, 5 τιμές για τη παράμετρο gamma και 4 τιμές για τη συνάρτηση kernel, όπως φαίνεται στον παρακάτω πίνακα.

Table 1: GridSearchCV parameters

C	Gamma	Kernel Function
0.01	Auto	Linear
0.1	Scale	RBF
1	0.1	Polynomial
10	1	Sigmoid
-	10	-

Γραφική αναπαράσταση μετρήσεων

Figure 1: Predict time of each model

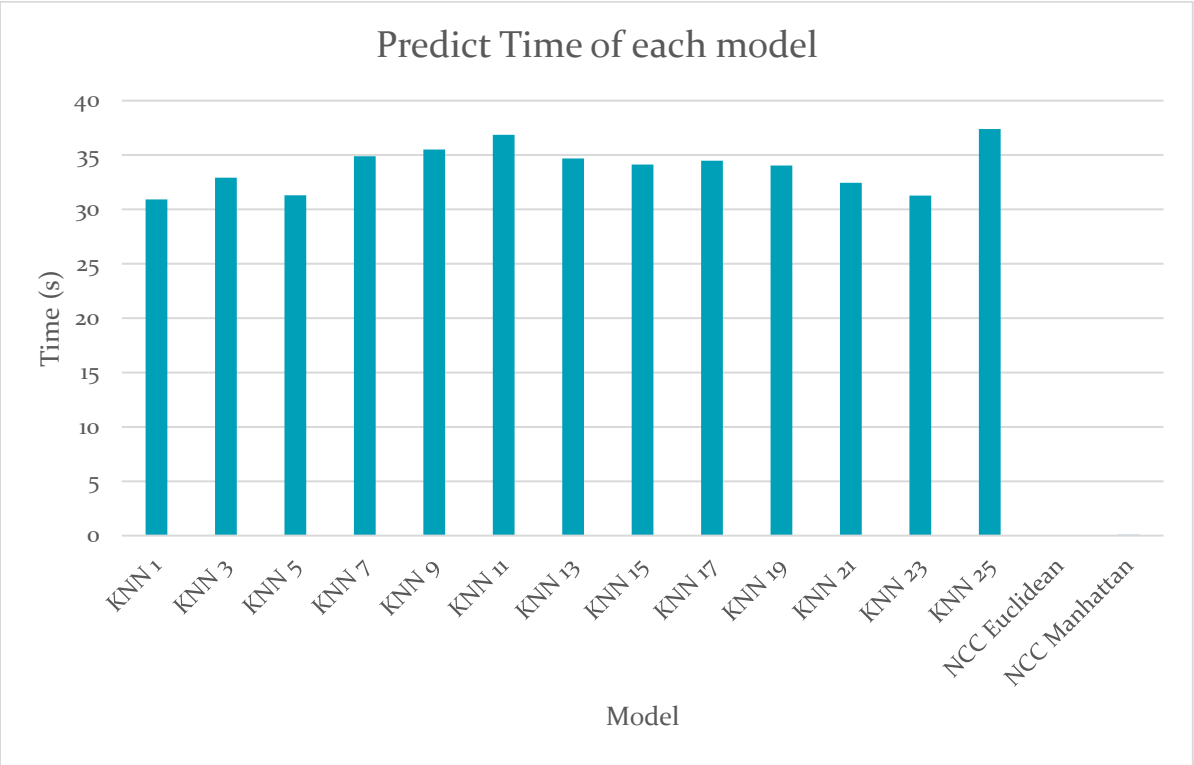


Figure 2: Fit time of each model

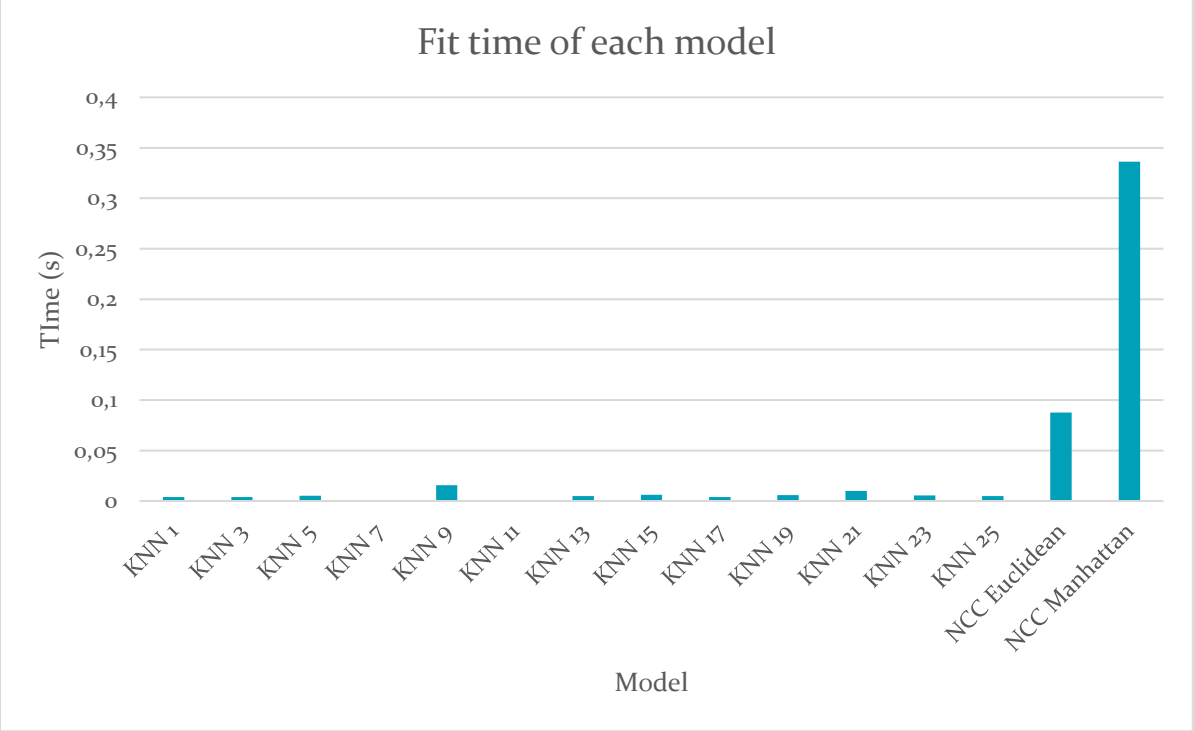


Figure 3: Total time of each model

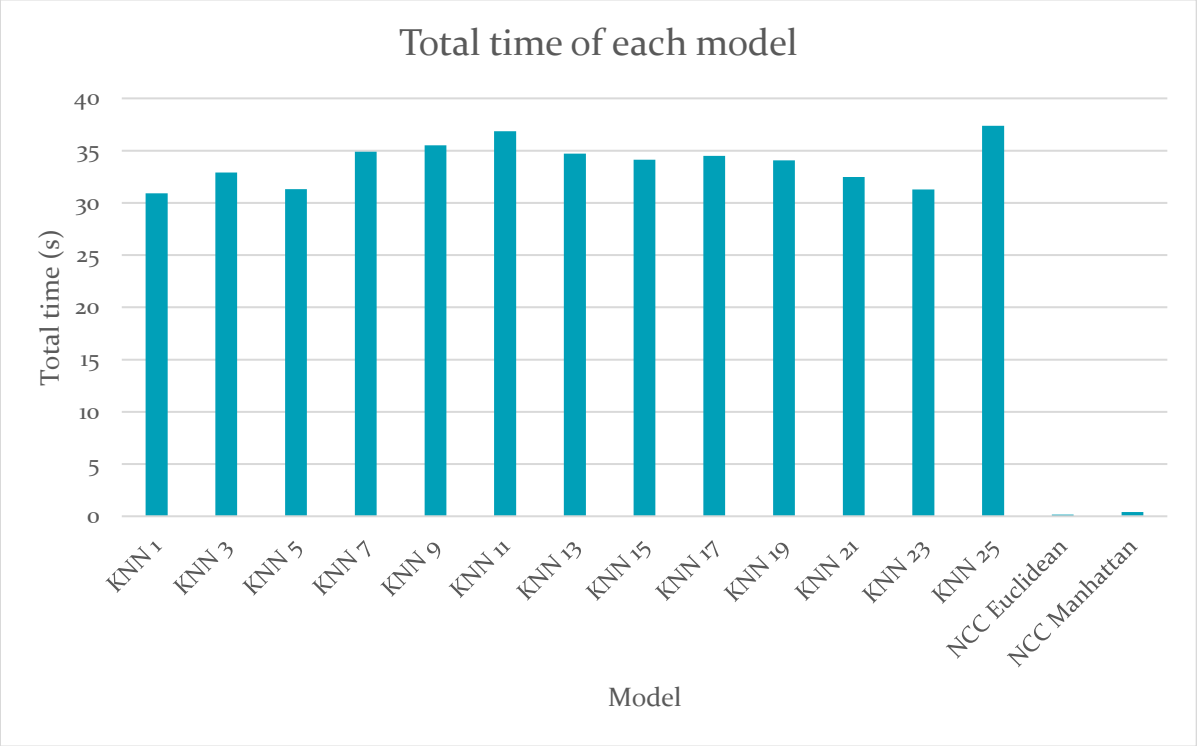


Figure 4: Accuracy percentage of KNN models

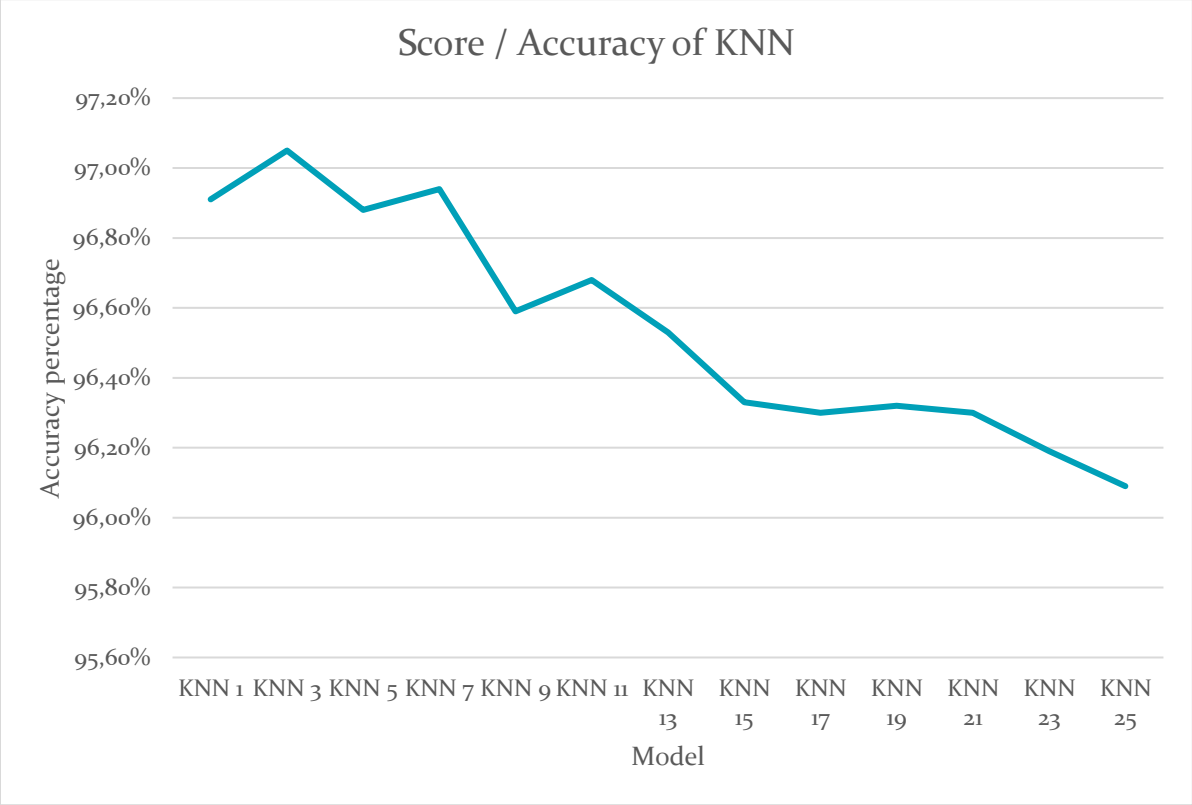


Figure 5: Accuracy percentage of each model

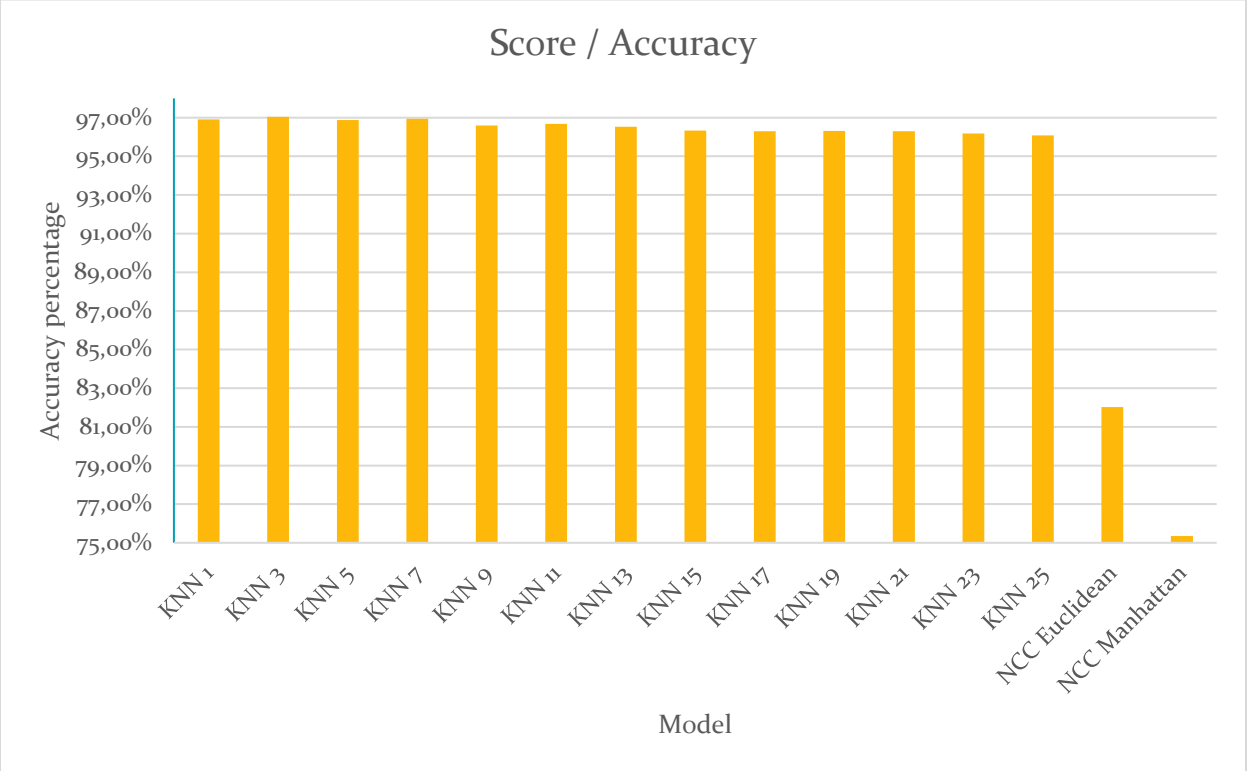


Figure 6: Accuracy percentage of KNN (3 neighbors) with Cross Validation

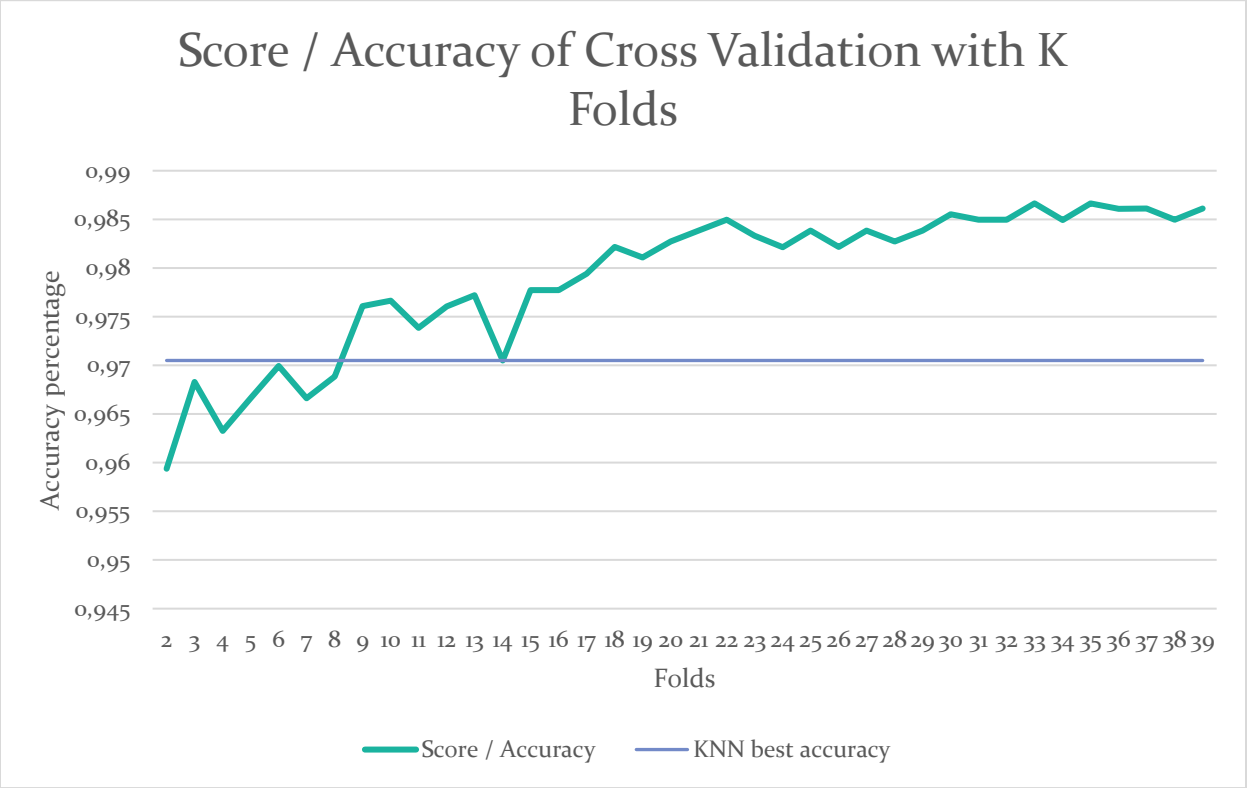


Figure 7: Best accuracy vs Kernel

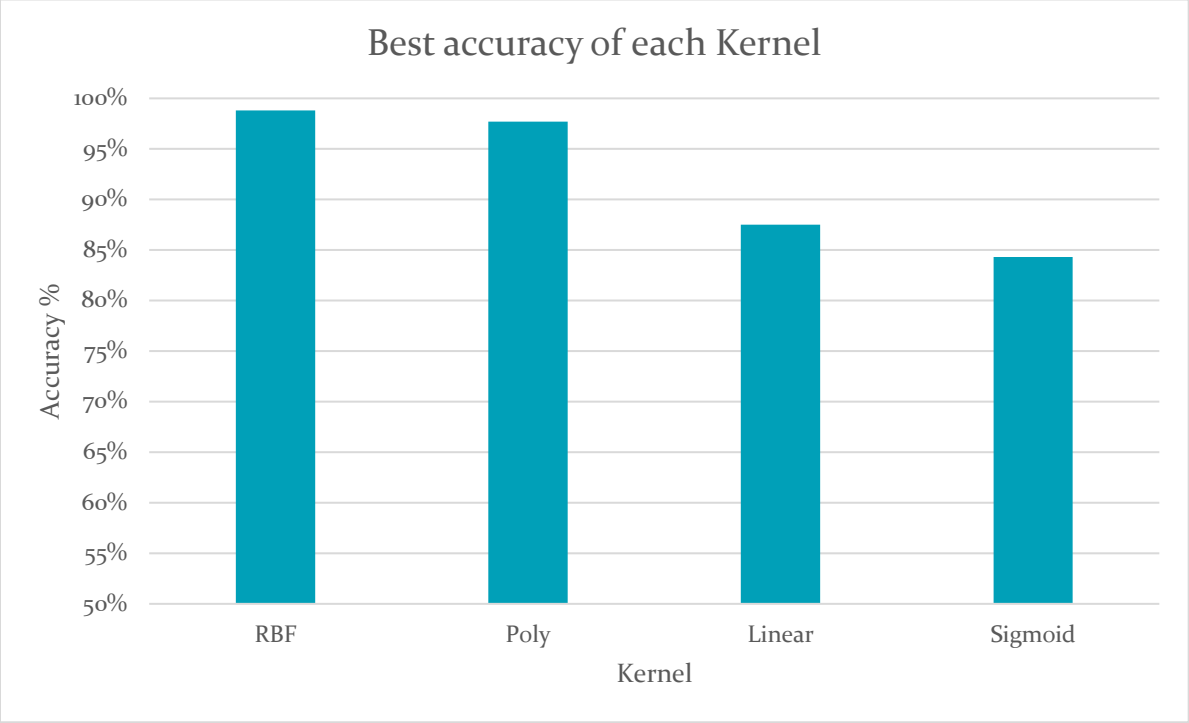
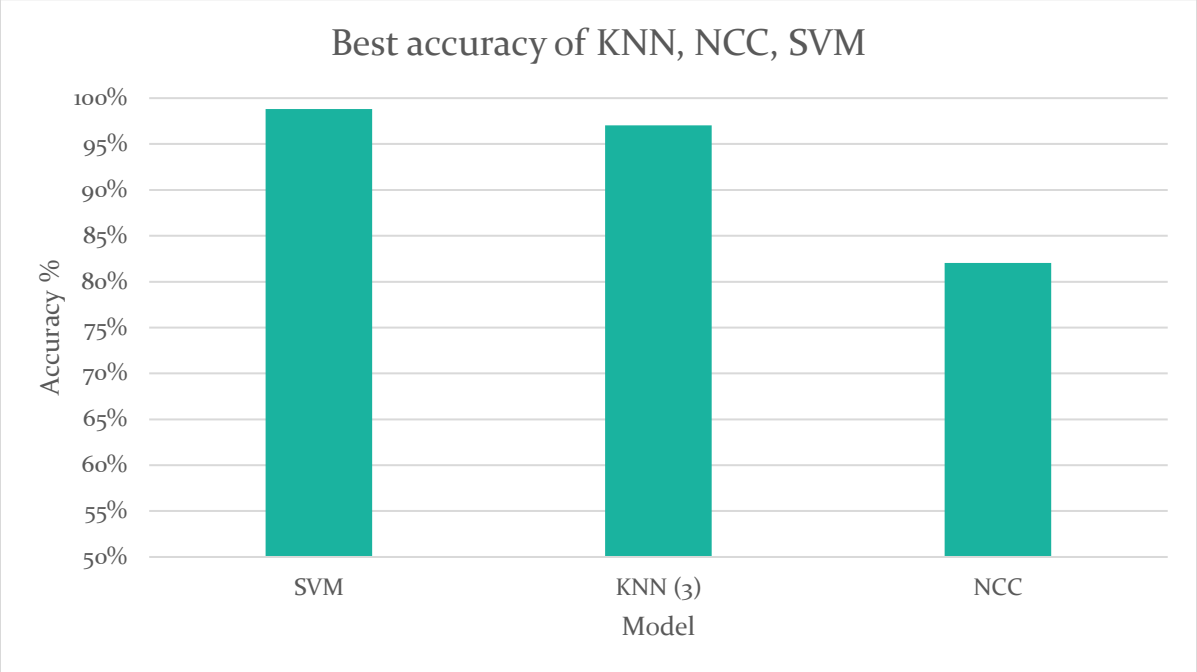


Figure 8: Best accuracy of KNN, NCC, SVM



Σχολιασμός Αποτελεσμάτων

KNN VS NCC ACCURACY

Συγκρίνοντας τα αποτελέσματα μεταξύ των KNN με διαφορετικό αριθμό γειτόνων, παρατηρούμε στο γράφημα 4 πως πιο ακριβής είναι αυτός με τους 3 γείτονες (97,05%). Ο συνολικός χρόνος κυμαίνεται από 31 μέχρι 37 δευτερόλεπτα, επομένως θεωρείται γενικά αργός αλγόριθμος. Να σημειωθεί πως ο χρόνος ποικίλλει ανάλογα με την επεξεργαστική ισχύ του μηχανήματος που χρησιμοποιούμε. Ο ίδιος κώδικας έτρεξε και σε διαφορετικό μηχάνημα και σε διαφορετικό περιβάλλον, και αυτό είχε σημαντική επιρροή στο χρόνο εκπαίδευσης και πρόβλεψης. Ωστόσο, όλες οι μετρήσεις που φαίνονται στους πίνακες του παραρτήματος Α λήφθηκαν ceteris paribus, και επομένως η σύγκριση μεταξύ των αποτελεσμάτων έχει νόημα.

Ο NCC από την άλλη παρατηρούμε πως είναι αρκετά γρήγορος. Στη περίπτωσή μας έχουμε κάνει reshape(flatten) το dataset από 2D σε 1D (28 * 28 to 784), επομένως έχει νόημα η ευκλείδεια απόσταση. [Γενικά δεδομένα σε 1D χρησιμοποιούνται σε πλήρως συνδεδεμένα νευρωνικά δίκτυα (όπως MLP), ενώ δεδομένα σε 2D χρησιμοποιούνται για συνελικτικά νευρωνικά δίκτυα.] Για σκοπούς παρατήρησης, υλοποιήθηκε ο NCC και με Manhattan απόσταση, ωστόσο αυτό είχε σημαντική επίδραση (μείωση) στην ακρίβειά του.

Παρατηρούμε επίσης στα γραφήματα 1 και 2 πως ο KNN έχει αρκετά μικρό χρόνο fit και πολύ μεγαλύτερο predict σε σύγκριση με τον NCC. Η μεγάλη διαφορά που βλέπουμε στους χρόνους μεταξύ KNN και NCC έγκειται στο γεγονός πως η περισσότερη δουλειά στον NCC γίνεται κατά τη διαδικασία fit, όπου ο αλγόριθμος υπολογίζει τα κέντρα (centroids) των κλάσεων, και επομένως στη συνέχεια απλώς συγκρίνει τις αποστάσεις αυτών από το σημείο που μας ενδιαφέρει. Δηλαδή στη συγκεκριμένη περίπτωση υλοποιεί 10 συγκρίσεις αφού έχουμε 10 κλάσεις. Ο KNN κάθε φορά ψάχνει τους πλησιέστερους K γείτονες και υπολογίζει πόσοι από αυτούς ανήκουν στην κάθε κλάση, ώστε να κατηγοριοποιήσει το σημείο που μελετάμε. Για το λόγο αυτό παρατηρούμε και μια μικρή αύξηση στο χρόνο όσο η παράμετρος K αυξάνεται.

Τελικά, ο NCC είναι σαφώς πιο γρήγορος από τον KNN. Ωστόσο, η ακρίβεια των προβλέψεων του NCC είναι αρκετά πιο χαμηλή (στην καλύτερη περίπτωση 82,03% έναντι 97,05% του KNN).

CROSS VALIDATION VS KNN ACCURACY

Χρησιμοποιήσαμε τη τεχνική Cross Validation στον KNN με 3 γείτονες καθώς αυτός είχε τη μεγαλύτερη ακρίβεια στη προηγούμενη σύγκριση. Βλέπουμε, από το γράφημα 6, πως η ακρίβεια αυξάνεται για μεγαλύτερα K-Folds, ενώ για K=35 φτάνουμε στην μέγιστη ακρίβεια που παρατηρήθηκε, 0,98664 ή 98,664%. Ακόμη όμως παρατηρούμε πως με τη τεχνική CV, οι περισσότερες μετρήσεις μας έχουν ακρίβεια μεγαλύτερη από 97,05%, η οποία ήταν η μέγιστη στο μοντέλο KNN χωρίς CV. Τελικά, συμπεραίνουμε πως με τη τεχνική CV, η ακρίβεια στα αποτελέσματά μας είναι σαφώς καλύτερη, ειδικά όσο αυξάνουμε τα folds.

SVM ACCURACY

Στο SVM παρατηρούμε ότι ο χρόνος που απαιτείται είναι εξαιρετικά μεγάλος για να κάνει fit το μοντέλο. Προσπαθήσαμε να τον μειώσουμε χρησιμοποιώντας PCA και scaling. Μέσω PCA, καταφέραμε να μειώσουμε τα components από 784 σε 87-236 ανάλογα με τη συνάρτηση kernel, κρατώντας το 90% της διαθέσιμης πληροφορίας, όπως ζητήθηκε.

Από τα αποτελέσματα που πήραμε, καταλήξαμε πως οι καλύτερες παράμετροι ήταν: C=10 & gamma="scale" για kernel="rbf". Επομένως ξανατρέξαμε τον αλγόριθμο για αυτό τον συνδυασμό παραμέτρων με μόνη kernel την RBF. Οι μέγιστες αποδόσεις που πήραμε από την Grid Search φαίνονται στο αντίστοιχο διάγραμμα. Τελικά καταφέραμε να φτάσουμε εξαιρετικά επίπεδα ακρίβειας μόνο όταν είχαμε 100% της διαθέσιμης πληροφορίας με kernel="rbf". Ωστόσο το μοντέλο χρειάστηκε αρκετό χρόνο για να κάνει fit. Η πολυωνυμική kernel ήταν η ταχύτερη, ωστόσο έχανε σε ακρίβεια από την rbf.

Επίσης η παράμετρος C επηρεάζει σημαντικά τον χρόνο fit & predict. Παρατηρούμε ότι για kernel="rbf" και gamma ="scale", με C=1 και το dataset περιορισμένο στο 20% του, το μοντέλο κάνει fit σε 184 seconds & predict σε 17 seconds, C=100 κάνει fit σε 1711 seconds & predict σε 74 seconds.

Η μέγιστη ακρίβεια βρέθηκε για τις εξής παραμέτρους:

- C = 10
- Gamma = scale
- Kernel = RBF

και ήταν test set accuracy = 98,8% & train set accuracy = 100%.

KNN, NCC, KNN CV VS SVM ACCURACY

Συγκρίνοντας όλους τους κατηγοριοποιητές, καταλήγουμε πως ο πιο ακριβής από όλους ήταν ο SVC. Ωστόσο ο χρόνος fit είναι αρκετά μικρότερος στο μοντέλο KNN, ενώ στο χρόνο predict κερδίζει και πάλι ο SVC. Ο NCC κερδίζει σε χρόνο predict και τους δύο προηγούμενους κατηγοριοποιητές, ωστόσο χάνει γενικά σε ακρίβεια.

ΓΕΝΙΚΟ ΣΥΜΠΕΡΑΣΜΑ

Τελικά, συμπεραίνουμε ότι το SVM μοντέλο, πετυχαίνει εξαιρετικά επίπεδα ακρίβειας, με σχετικά μικρό χρόνο predict, παρόλο που χρειάζεται λίγο παραπάνω χρόνο για να κάνει fit Ο NCC είναι εξαιρετικά γρήγορος στο να κάνει predict, ωστόσο δεν έχει τόσο καλή ακρίβεια όσο άλλα μοντέλα. Καταλήγουμε πως το SVM μοντέλο είναι το καλύτερο.

Παράρτημα

(Α) ΠΙΝΑΚΕΣ ΜΕΤΡΗΣΕΩΝ

Table 2: Μετρήσεις fit time και predict time KNN & NCC

Model	Fit time (s)	Predict time (s)
KNN 1	0,004010677337646484	30,921151638031006
KNN 3	0,003977298736572266	32,90735363960266
KNN 5	0,0053746700286865234	31,2930598258972l7
KNN 7	0,0	34,9021737575531
KNN 9	0,01564168930053711	35,497914791107l8
KNN 11	0,0	36,85045766830444
KNN 13	0,004988431930541992	34,698371887207O3
KNN 15	0,006287336349487305	34,1180405616760l5
KNN 17	0,003983736038208008	34,49005627632141
KNN 19	0,0057260990142822266	34,0493946075439l5
KNN 21	0,010051488876342773	32,45123600959778
KNN 23	0,0055880546569824l2	31,26777744293213
KNN 25	0,004971265792846l8	37,37762904167175
NCC Euclidean	0,08776473999023438	0,06194138526916504
NCC Manhattan	0,3364279270172119	0,07768750190734863

Table 3: Μετρήσεις συνολικού χρόνου και ακρίβειας KNN & NCC

Model	Total time (s)	Score / Accuracy
KNN 1	30,9251623153686 52	0,9691 or 96,91%
KNN 3	32,91133094	0,9705 or 97,05%
KNN 5	31,2984345	0,9688 or 96,88%
KNN 7	34,90217376	0,9694 or 96,94%
KNN 9	35,51355648	0,9659 or 96,59%
KNN 11	36,85045767	0,9668 or 96,68%
KNN 13	34,70336032	0,9653 or 96,53%
KNN 15	34,1243279	0,9633 or 96,33%
KNN 17	34,49404001	0,963 or 96,3%
KNN 19	34,05512071	0,9632 or 96,32%
KNN 21	32,4612875	0,963 or 96,3%
KNN 23	31,2733655	0,9619 or 96,19%
KNN 25	37,3826003074646	0,9609 or 96,09%
NCC Euclidean	0,14970612525939942	0,8203 or 82,03%
NCC Manhattan	0,414115428924560 53	0,7535 or 75,35%

Table 4: Μετρήσεις ακρίβειας τεχνικής Cross Validation σε KNN Classifier με 3 γείτονες

Folds	Score / Accuracy
2	0,9593788941437034
3	0,9682804674457429
4	0,9632764167285326
5	0,966621788919839
6	0,9699498327759198
7	0,9666155676764869
8	0,9688492063492065
9	0,9760831937465103
10	0,9766325263811299
11	0,9738651667052085
12	0,9760700969425803
13	0,9771941644009209
14	0,9705019725913621
15	0,9777170868347338
16	0,9777328934892543
17	0,9794091221394217
18	0,982182940516274
19	0,9810868155831909
20	0,9827340823970039
21	0,9838512149045665
22	0,9849648243957188

23	0,983312871315073
24	0,9821471471471472
25	0,9838341158059468
26	0,9821866539257845
27	0,9838350335862772
28	0,9827352335164835
29	0,9838618501431463
30	0,9855084745762712
31	0,9849735573639326
32	0,9849819862155391
33	0,9866340169370473
34	0,9849419448476053
35	0,9866408101702218
36	0,986077097505669
37	0,9861072807501378
38	0,9849640724150803
39	0,9861120994330986

(Β) ΚΩΔΙΚΑΣ

KNN & NCC

```python
# import the libraries
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from time import time
import numpy as np

# load the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
print(f"Train shape: {x_train.shape} ")
print(f"Test shape: {x_test.shape} ")

# reshape model to work with (flatten it into 1D)
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)
print(f"Reshaped train shape: {x_train.shape} ")
print(f"Reshaped test shape: {x_test.shape} ")

```

```python
# KNN (for K values: 1 to 25)
kValues = np.arange(1, 27, 2)
scores = []
times = []
for i in kValues:
    knn = KNeighborsClassifier(n_neighbors=i)
    startKNNFitTime = time()
    knn.fit(x_train, y_train)
    fitKNNTime = time()-startKNNFitTime
    accuracyKNN = knn.score(x_test, y_test)
    startKNNPredictTime = time()
    knn.predict(x_test)
    predictKNNTime = time()-startKNNPredictTime
    totalKNNTime = fitKNNTime + predictKNNTime
    times.append(totalKNNTime)
    scores.append(accuracyKNN)
    print(f"KNN with {i} neighbor(s) fit time: {fitKNNTime}s \n"
          f"KNN with {i} neighbor(s) predict time: {predictKNNTime}s \n"
          f"KNN with {i} neighbor(s) accuracy: {accuracyKNN} \n")
    print(times)
    print(scores)
```

```python
# NCC (for distances: Euclidean, Manhattan)
distances = ["manhattan", "euclidean"]
for distance in distances:
    nc = NearestCentroid(distance)
    startNCCFitTime = time()
    nc.fit(x_train, y_train)
    fitNCCTime = time()-startNCCFitTime
    accuracyNCC = nc.score(x_test, y_test)
    startNCCPredictTime = time()
    nc.predict(x_test)
    predictNCCTime = time()-startNCCPredictTime
    totalKNNTime = fitNCCTime + predictNCCTime
    print(f"NCC {distance} fit time: {fitNCCTime}s \n"
          f"NCC {distance} predict time: {predictNCCTime}s \n"
          f"NCC {distance} accuracy: {accuracyNCC} \n")
```

KNN Cross Validation

```python
# import the libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, cross_val_score

# load the dataset
digits = load_digits()

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target)

# for 3 neighbors in KNN, for K fold values 2-40 print the mean accuracy
for i in range(2, 40):
    print(sum(cross_val_score(KNeighborsClassifier(n_neighbors=3), digits.data, digits.target, cv=i))/i)
```

SVM

```python
import time
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# load the dataset
mnist = tf.keras.datasets.mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(f"Data Loaded")

X_train = X_train.astype(float) / 255.0
X_test = X_test.astype(float) / 255.0

# reshape dataset
X_train = X_train.reshape((X_train.shape[0], -1), order="F")
X_test = X_test.reshape((X_test.shape[0], -1), order="F")
print(f"Data Reshaped")
```

```python
# standardize dataset
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(f"Data Standardized")

# change labels to even 0 and odd 1
y_train = y_train % 2
y_test = y_test % 2

# PCA
pca = PCA(.90)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)
print(f"PCA is complete with {pca.n_components_} components")
```

```python
# SVM hyperparameters for tuning
param_grid = {
    "C": [0.01, 0.1, 1, 10],
    "gamma": ['auto', 'scale', 0.1, 1, 10],
    "kernel": ["rbf", "poly", "linear", "sigmoid"]
}

# perfrom grid search cv
grid = GridSearchCV(SVC(), param_grid, verbose=2, n_jobs=-1)
grid.fit(X_train, y_train)
```

```python
# SVM
startTime = time()
svm = SVC(kernel='rbf', C=100, gamma=0.01)
svm.fit(X_train, y_train)
fitTime = time() - startTime
print(f"Fit time of SVM is: {fitTime}s")
startTimePredict = time()
y_pred = svm.predict(X_test)
predictTime = time() - startTimePredict
print(f"Predict time of SVM is: {predictTime}s")
predicted = svm.predict(X_train)
print("accuracy of test set:", metrics.accuracy_score(y_test, y_pred))
print("accuracy of train set:", metrics.accuracy_score(y_train, predicted))
```

```
C:\Users\conmy\pycharmProjects\venv\Scripts\pythor
2022-12-22 15:41:03.503144: W tensorflow/stream_ex
2022-12-22 15:41:03.503478: I tensorflow/stream_ex
Data Loaded
Data Reshaped
Data Standardized
PCA is complete with 236 components
Fit time of SVM is: 1711.8074061870575s
Predict time of SVM is: 74.13484644889832s
accuracy of test set: 0.9668
accuracy of train set: 1.0

Process finished with exit code 0
```

```
C:\Users\conmy\pycharmProjects\venv\Scripts\py
2022-12-21 13:50:41.973978: W tensorflow/strea
2022-12-21 13:50:41.974898: I tensorflow/strea
Number of components: 236
PCA is complete
Fit time of SVM is: 211.73542141914368s
Predict time of SVM is: 34.14662289619446s
accuracy of test set: 0.9756
accuracy of train set: 0.9982333333333333

Process finished with exit code 0
```

Smaller scale Grid Search Results

```
C:\Users\conmy\pycharmProjects\venv\Scripts\python.exe C:\Users\conmy\pycharmProj
2022-12-23 02:16:58.333057: W tensorflow/stream_executor/platform/default/dso_loa
2022-12-23 02:16:58.333345: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Data Loaded
Data Reshaped
Data Standardized
PCA is complete with 236 components
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[CV] END .....................C=1, gamma=scale, kernel=rbf; total time= 9.1min
[CV] END .....................C=1, gamma=scale, kernel=rbf; total time= 9.3min
[CV] END .....................C=1, gamma=scale, kernel=rbf; total time= 9.5min
[CV] END .....................C=1, gamma=scale, kernel=rbf; total time= 9.6min
[CV] END .....................C=1, gamma=scale, kernel=rbf; total time= 9.7min
[CV] END .....................C=1, gamma=scale, kernel=poly; total time=10.7min
[CV] END .....................C=1, gamma=scale, kernel=poly; total time=10.8min
[CV] END .....................C=1, gamma=scale, kernel=poly; total time=10.9min
[CV] END .....................C=1, gamma=scale, kernel=poly; total time=11.1min
[CV] END .....................C=1, gamma=scale, kernel=poly; total time=12.2min
[CV] END .....................C=1, gamma=0.1, kernel=poly; total time=13.6min
[CV] END .....................C=10, gamma=scale, kernel=rbf; total time=12.4min
```

```
Best parameters set found: {'C': 10, 'gamma': 'scale', 'kernel': 'poly'}
Best score found: 0.9834166666666668
Grid scores:
0.978 (+/-0.002) for {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'} in 502.407 (+/-24.064)
0.978 (+/-0.001) for {'C': 1, 'gamma': 'scale', 'kernel': 'poly'} in 596.654 (+/-65.782)
0.603 (+/-0.007) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'} in 4294.827 (+/-23.857)
0.981 (+/-0.001) for {'C': 1, 'gamma': 0.1, 'kernel': 'poly'} in 842.706 (+/-169.657)
0.983 (+/-0.002) for {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'} in 702.708 (+/-41.277)
0.983 (+/-0.001) for {'C': 10, 'gamma': 'scale', 'kernel': 'poly'} in 758.146 (+/-156.984)
0.614 (+/-0.008) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'} in 5528.840 (+/-181.903)
0.981 (+/-0.001) for {'C': 10, 'gamma': 0.1, 'kernel': 'poly'} in 786.345 (+/-202.749)

Process finished with exit code 0
```

Βιβλιογραφία

1. “Principal Component Analysis with Python - GeeksforGeeks.” Principal Component Analysis with Python, https://www.geeksforgeeks.org/principal-component-analysis-with-python/.
2. “Major Kernel Functions in Support Vector Machine (SVM).” Major Kernel Functions in Support Vector Machine SVM, 7 Feb. 2022, https://www.geeksforgeeks.org/major-kernel-functions-in-support-vector-machine-svm/.
3. Singh, Rana. Principal Component Analysis(Pca) with Code on Mnist Dataset. Analytics Vidhya, 19 Sept. 2019, https://medium.com/analytics-vidhya/principal-component-analysis-pca-with-code-on-mnist-dataset-da7de0d07c22.
4. “Dhairya Kothari.” SVM with MNIST, https://dmkothari.github.io/Machine-Learning-Projects/SVM_with_MNIST.html.
5. “SVM Hyperparameter Tuning Using GRIDSEARCHCV.” GeeksforGeeks, 23 Aug. 2022, https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/.
6. “API Reference.” Scikit, https://scikit-learn.org/stable/modules/classes.html.
7. Maklin, Cory. “Support Vector Machine Python .” Medium, Towards Data Science, 10 May 2022, https://towardsdatascience.com/support-vector-machine-python-example-d67d9b63f1c8.
8. Galarnyk, Michael. “PCA Using Python (Scikit-Learn).” Medium, Towards Data Science, 29 Nov. 2022, https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60.