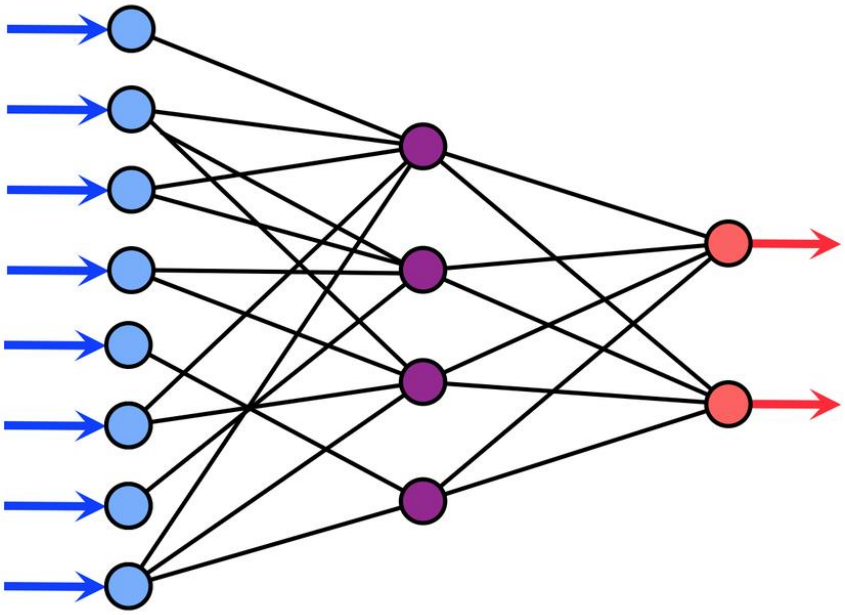
Εργασία 4 στην Αναγνώριση προτύπων



Χατζησάββας Χρήστος

Ακαδημαϊκό έτος 2023-2024

ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ
ΤΜΗΜΑ ΗΜ&ΜΥ

Άσκηση 1:

Στο πλαίσιο της άσκησης αυτής, θα χρησιμοποιηθούν για τα νευρωνικά δίκτυα που θα σχεδιαστούν οι συναρτήσεις ενεργοποίησης Sigmoid και ReLu. Αναφορικά με την Sigmoid, αποτελεί μια συνάρτηση κατάλληλη για προβλήματα ταξινόμησης μεταξύ 2 κλάσεων καθώς μπορεί να δώσει πραγματικές τιμές στις πιθανότητες. Η έξοδος της συχνά είναι φραγμένη ώστε να μην υπάρξουν πολύ μεγάλες τιμές εξόδου κατά την εκπαίδευση. Το κύριο μειονέκτημα της είναι το vanishing gradient, όπου υπάρχει κορεσμός σε μικρές τιμές για υπερβολικά μεγάλη είσοδο περιορίζοντας, έτσι, την ροή της κλίσης προς τα πίσω μέσα από τα διάφορα layers. Επίσης, ο υπολογισμός του εκθετικού μέρους που περιέχει, είναι υπολογιστικά δαπανηρός και γι' αυτό αποφεύγεται ειδικά σε μεγάλα dataset. Από την άλλη πλευρά, η ReLu προσπερνά αυτό το πρόβλημα της σιγμοειδούς συνάρτησης, καθώς είναι πιο απλή υπολογιστικά. Το μεγαλύτερο της πρόβλημα ωστόσο, είναι ότι λόγω της κατασκευής της, αν τα βάρη οδηγούν σε αρνητικές εξόδους στον νευρώνα τότε η συνάρτηση ενεργοποίησης θα επιστρέφει πάντα μηδέν.

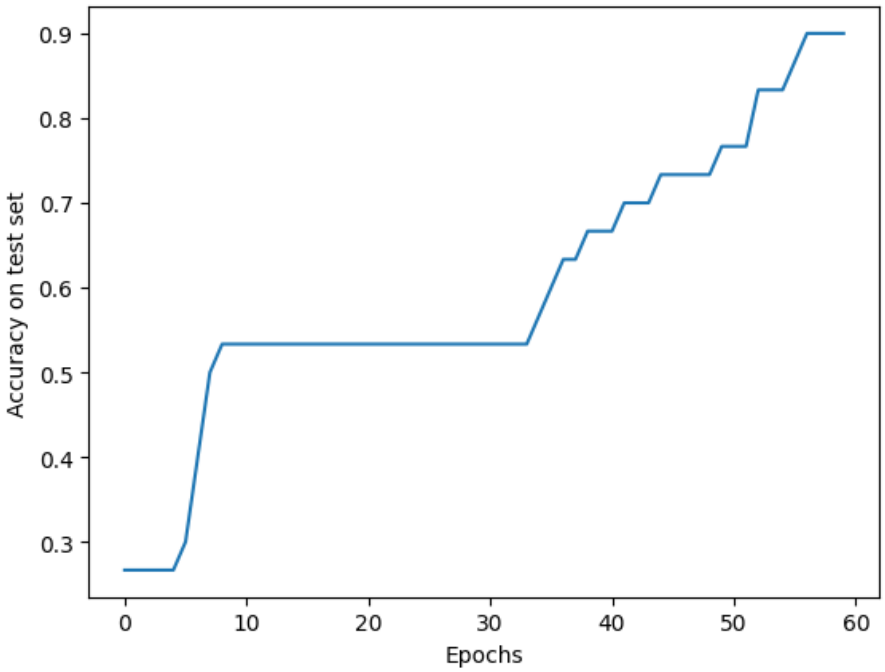
A) Χωρίζουμε το dataset σε train και test set με την χρήση της βιβλιοθήκης sklearn και πιο συγκεκριμένα με την συνάρτηση train_test_split. Ρυθμίζουμε την παράμετρο test_size = 0.2 ώστε το 20% του dataset να χρησιμοποιηθεί για testing.

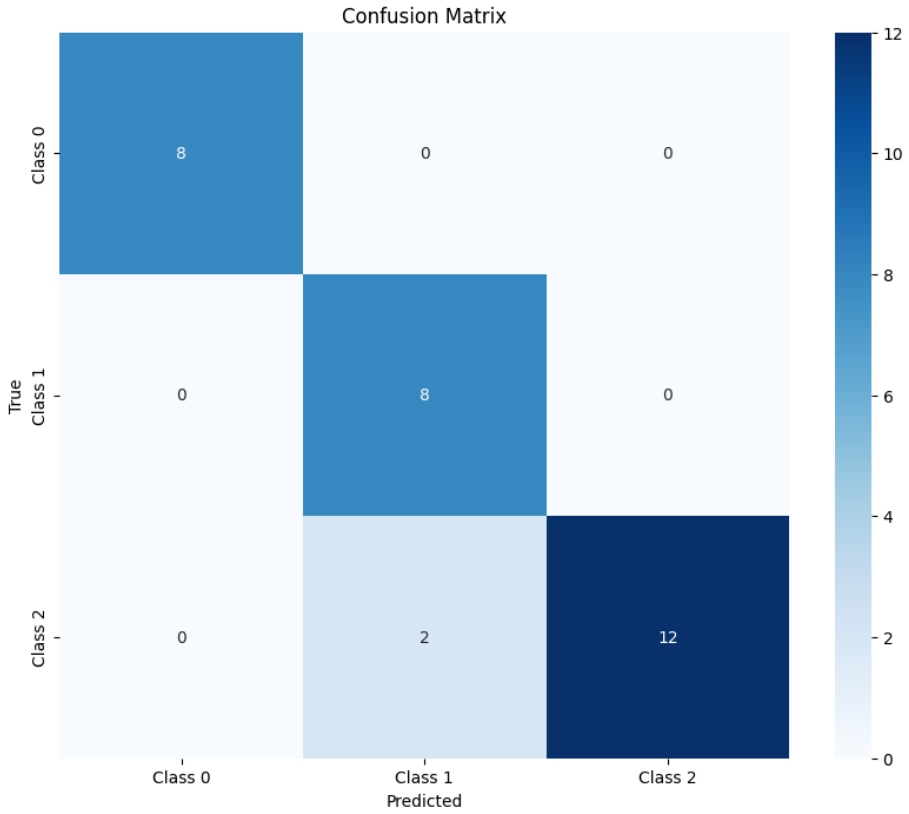
B) Πριν ξεκινήσουμε την δόμηση του δικτύου πρέπει να φέρουμε τα data σε μορφή που είναι αποδεκτή από την βιβλιοθήκη PyTorch. Πιο συγκεκριμένα, τα δεδομένα μετατρέπονται σε tensors και με την εντολή TensorDataset τα ομαδοποιούμε σε train set και test set ώστε να περιέχουν δείγματα και ετικέτες κλάσεων. Για τη δημιουργία του νευρωνικού δικτύου, χρησιμοποιούμε μια κλάση με όνομα NeuralNetwork βάζοντας ως όρισμα nn.Module για να δηλωθεί πως πρόκειται για νευρωνικό δίκτυο. Στη συνέχεια, ορίζεται η συνάρτηση init η οποία υποδεικνύει από τι αποτελείται το δίκτυο. Για το ερώτημα αυτό δημιουργούνται 2 layers . Στο πρώτο layer, έχουμε 4 εισόδους (προκύπτει από τον αριθμό των χαρακτηριστικών) και 30 νευρώνες. Στην έξοδο του layer αυτού εφαρμόζεται η σιγμοειδής συνάρτηση το αποτέλεσμα της οποίας τροφοδοτεί το δεύτερο layer. Στο δεύτερο layer έχουμε 3 νευρώνες που παράγουν και τις επιθυμητές εξόδους (όσες και ο αριθμός των κλάσεων). Τέλος, ορίζεται η συνάρτηση forward όπου εισάγεται το tensor x μέσα στο δίκτυο που δημιουργήθηκε και επιστρέφεται το αποτέλεσμα.

Απαραίτητα βήματα πριν την εκπαίδευση του μοντέλου είναι ο ορισμός κάποιων παραμέτρων (hyperparameters). Συγκεκριμένα, γίνεται καθορισμός του learning rate, της συνάρτησης σφάλματος και του optimizer. Επιλέγονται learning_rate = 0.1, ως loss function η CrossEntropy μιας και έχουμε classification problem και ως optimizer επιλέγεται ο SGD.

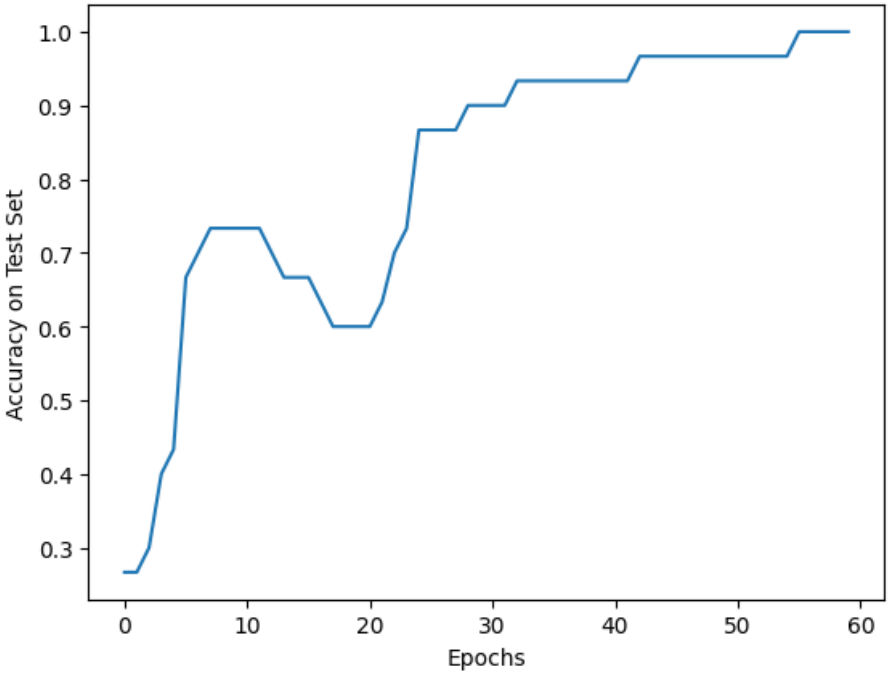
Έπειτα από όλα αυτά, ορίζονται 2 συναρτήσεις για την εκπαίδευση και το test του δικτύου (train_loop και test_loop). Η train loop υπολογίζει για το train_dataloader τα predictions και τα loss. Στη συνέχεια, γίνεται back propagation και μέσω του optimizer.zero_grad μηδενίζεται ο optimizer για να μην κρατήσει τίποτα από προηγούμενη εκτέλεση. Για την διάδοση του σφάλματος προς τα πίσω χρησιμοποιούμε το loss.backward και για να κάνει update τα βάρη χρησιμοποιούμε το optimizer.step. Η test loop αξιοποιεί το test_dataloader και υπολογίζει σε κάθε epoch πόσα σωστά predictions έγιναν αφού επιστρέφει την ακρίβεια και το μέσο σφάλμα.

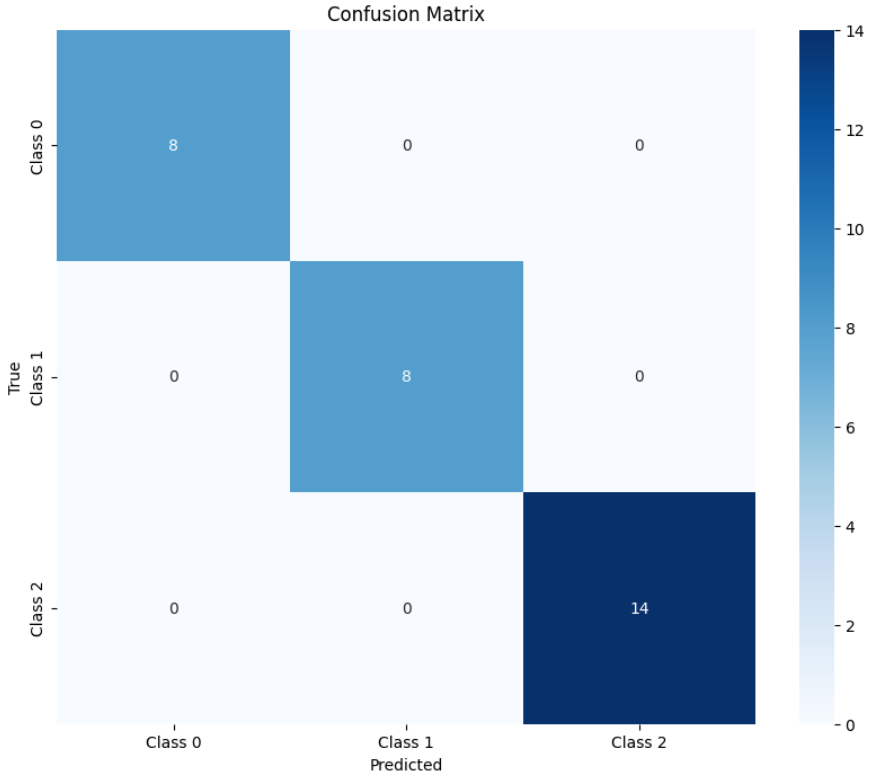
Γ) Η εξέλιξη του test accuracy και ο πίνακας σύγχυσης για το test set φαίνονται παρακάτω. Παρατηρούμε πως το μοντέλο φτάνει κοντά στο 90% στο τέλος των εποχών. Γενικά στην ανάπτυξη νευρωνικών δικτύων οι πολλές εποχές κοστίζουν υπολογιστικά γι' αυτό και αποφεύγονται. Η συνάρτηση sigmoid όπως προαναφέρθηκε δεν αποτελεί ιδανική επιλογή για multiclass classification πρόβλημα. Παρόλα αυτά, λόγω της απλότητας του προβλήματος και του μικρού dataset κατορθώνει αρκετά καλά αποτελέσματα.



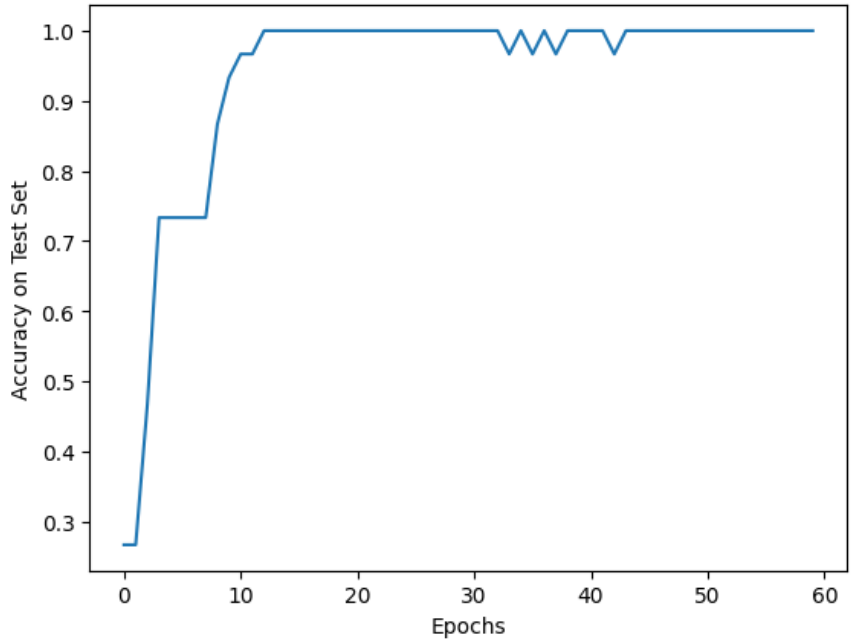


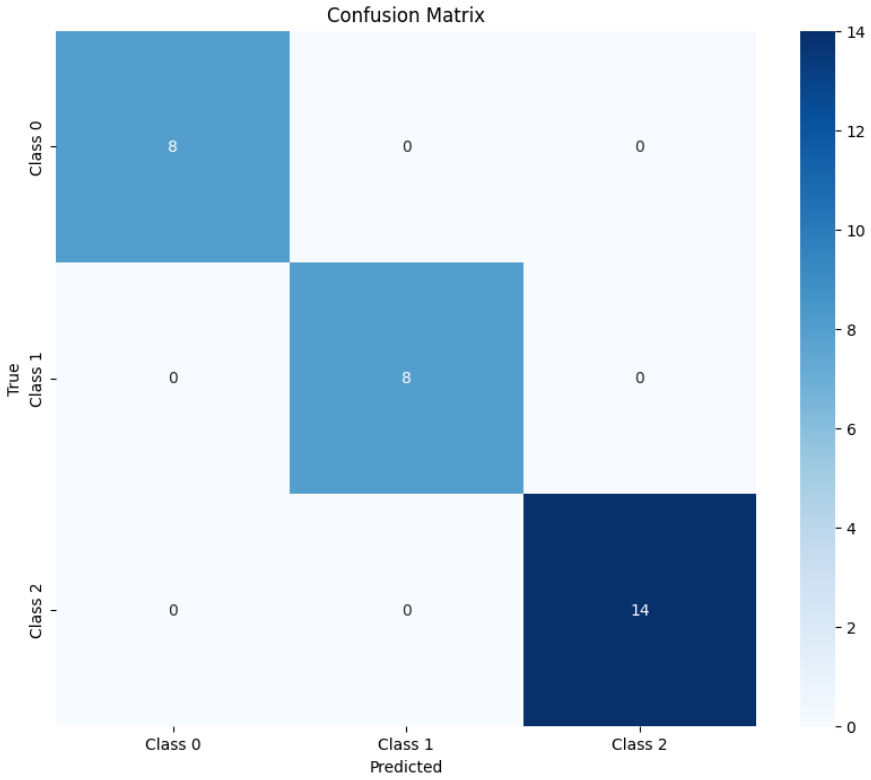
Δ) Για το ερώτημα αυτό χρησιμοποιούμε τον ίδιο κώδικα με παραπάνω απλά μέσα στο μοντέλο αλλάζουμε την συναρτήση ενεργοποίησης από sigmoid σε ReLu. Γενικά η ReLu είναι πιο απλή υπολογιστικά και δίνει λύσεις σε προβλήματα που έχει η σιγμοειδής. Σε αυτήν την εκτέλεση παρατηρούμε πως το μοντέλο κατορθώνει να φτάσει ακρίβεια 100% στις 60 εποχές που έτρεξε. Η διαφορά με την περίπτωση που χρησιμοποιούμε sigmoid συνάρτηση ενεργοποίησης είναι προφανής καθώς το μοντέλο μπορεί να οδηγηθεί σε καλύτερα αποτελέσματα. Στα νευρωνικά δίκτυα με πολλαπλές κλάσεις εκτός από την ReLu χρησιμοποιούνται και παραλλαγές της Το διάγραμμα του accuracy και ο πίνακας σύγχυσης ακολουθούν.





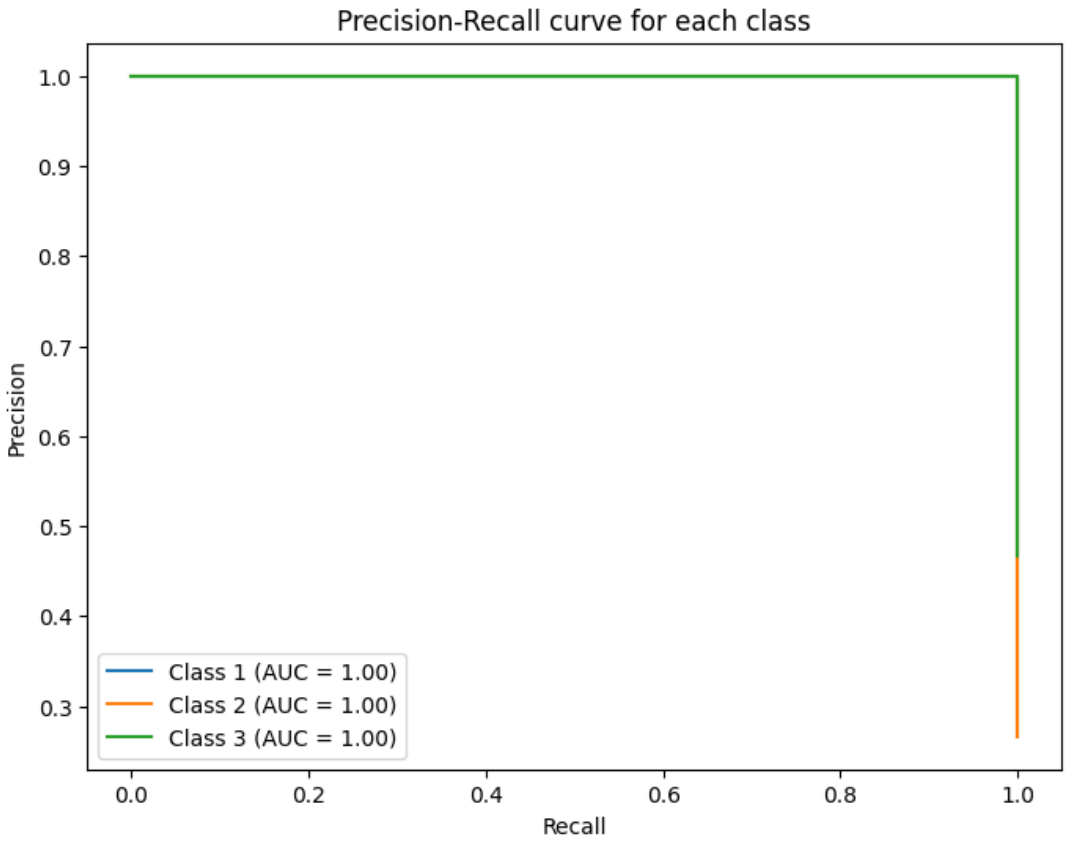
E) Στο ερώτημα αυτό προκειμένου να βελτιώσουμε το μοντέλο και να καταφέρουμε να το κάνουμε να συγκλίνει πιο γρήγορα στην μέγιστη δυνατή ακρίβεια έγιναν κάποιες αλλαγές. Πιο συγκεκριμένα, ορίσθηκε το learning rate σε 0.01 και επιλέχθηκε ως optimizer ο Adam σε σχέση με τα προηγούμενα ερωτήματα που ήταν ο SGD. Όσον αφορά την αρχιτεκτονική του δικτύου προστέθηκε ένα ακόμη layer με αποτέλεσμα το δίκτυο να έχει 3 layers. Τα αποτελέσματα είναι εμφανώς καλύτερα αφού παρατηρούμε πως το μοντέλο συγκλίνει πολύ γρήγορα στο 100%. Επειδή το Iris Dataset είναι πολύ μικρό dataset με μόλις 3 κλάσεις δεν υπάρχει ιδιαίτερη δυσκολία στην επίτευξη μεγάλης ακρίβειας. Το διάγραμμα της ακρίβειας συναρτήσει των εποχών ακολουθεί.





Στο confusion matrix, όπως είναι λογικό δεν παρατηρούμε λανθασμένες προβλέψεις καθώς η ακρίβεια του μοντέλου αγγίζει το 100% όπως και στην περίπτωση που χρησιμοποιήθηκε απλά η ReLu.

ΣΤ) Ως καλύτερο μοντέλο για το ερώτημα αυτό, είναι το μοντέλο του προηγούμενου ερωτήματος καθώς επιτυγχάνει μέγιστη απόδοση και συγκλίνει γρήγορα. Δεδομένου ότι το μοντέλο φτάνει 100% ακρίβεια στο test set, οδηγεί στο συμπέρασμα πως το νευρωνικό δίκτυο εκτελεί βέλτιστη ταξινόμηση. Απόδειξη της παραπάνω παρατήρησης αποτελούν οι καμπύλες του ακόλουθου διαγράμματος precision-recall αφού με βάση την τιμή του AUC έχουμε ιδανική συμπεριφορά στον ταξινομητή.



Άσκηση 2:

Οι auto-encoders αποτελούν ένα νευρωνικό δίκτυο το οποίο δημιουργεί μια μη γραμμική αντιστοίχιση των δεδομένων εισόδου σε έναν καινούργιο χώρο χαρακτηριστικών. Αποτελούνται από το κομμάτι του κωδικοποιητή (encoder) και του αποκωδικοποιητή (decoder). Στόχος του encoder είναι να βρει μια βέλτιστη αναπαράσταση σε χαμηλότερες διαστάσεις για τα δεδομένα που τροφοδοτούνται στο δίκτυο. Αρχιτεκτονικά, αυτό υλοποιείται από διαδοχικά layers όπου κάθε νέο layer περιέχει λιγότερους νευρώνες μειώνοντας έτσι τις διαστάσεις των δεδομένων. Αντίθετα ο decoder, λαμβάνει ως είσοδο το αποτέλεσμα του encoder και προσπαθεί να επανασυνθέσει την συμπιεσμένη πληροφορία προσθέτοντας layer συμμετρικά, ενώ παράλληλα αυξάνει το πλήθος των νευρώνων.

Α) Για την επίλυση της συγκεκριμένης άσκησης έχουμε ορίσει ως συνάρτηση σφάλματος το μέσο τετραγωνικό σφάλμα (MSE). Συνεπώς, στην εκτέλεση του κώδικα σε κάθε epoch λαμβάνουμε το επιθυμητό είδος σφάλματος. Συγκρίνοντας τον κώδικα του εργαστηρίου με αυτόν της άσκησης, παρατηρούμε ότι μικρότερο μέσο τετραγωνικό σφάλμα επιτυγχάνει ο κώδικας του εργαστηρίου. Αυτό συμβαίνει καθώς το δίκτυο που σχεδιάστηκε στον κώδικα αυτόν, είναι πιο απλό (έχει μόνο 2 layers), είναι πιο γενικευμένο και μειώνει τον κίνδυνο για overfitting. Από την άλλη πλευρά, ο κώδικας της άσκησης αυτής έχει περισσότερα layers και οδηγεί σε bottleneck στο latent space αφού από την 28x28 εικόνα φτάνει σε ένα latent space 3 διαστάσεων.

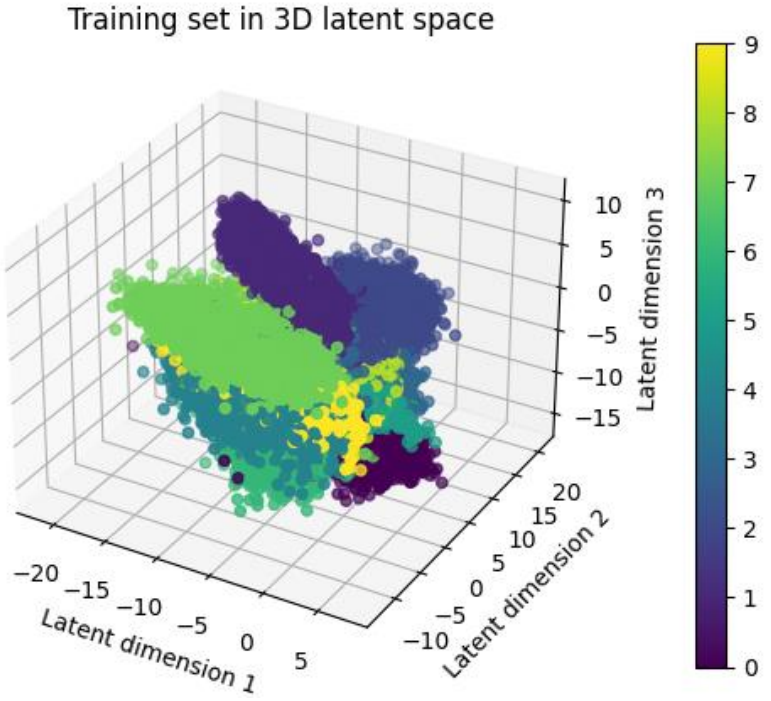
Autoencoder εργαστηρίου

```
Epoch [1/20], Loss: 0.0172
Epoch [2/20], Loss: 0.0100
Epoch [3/20], Loss: 0.0101
Epoch [4/20], Loss: 0.0078
Epoch [5/20], Loss: 0.0074
Epoch [6/20], Loss: 0.0087
Epoch [7/20], Loss: 0.0089
Epoch [8/20], Loss: 0.0083
Epoch [9/20], Loss: 0.0073
Epoch [10/20], Loss: 0.0083
Epoch [11/20], Loss: 0.0087
Epoch [12/20], Loss: 0.0066
Epoch [13/20], Loss: 0.0058
Epoch [14/20], Loss: 0.0063
Epoch [15/20], Loss: 0.0066
Epoch [16/20], Loss: 0.0056
Epoch [17/20], Loss: 0.0067
Epoch [18/20], Loss: 0.0060
Epoch [19/20], Loss: 0.0057
Epoch [20/20], Loss: 0.0066
```

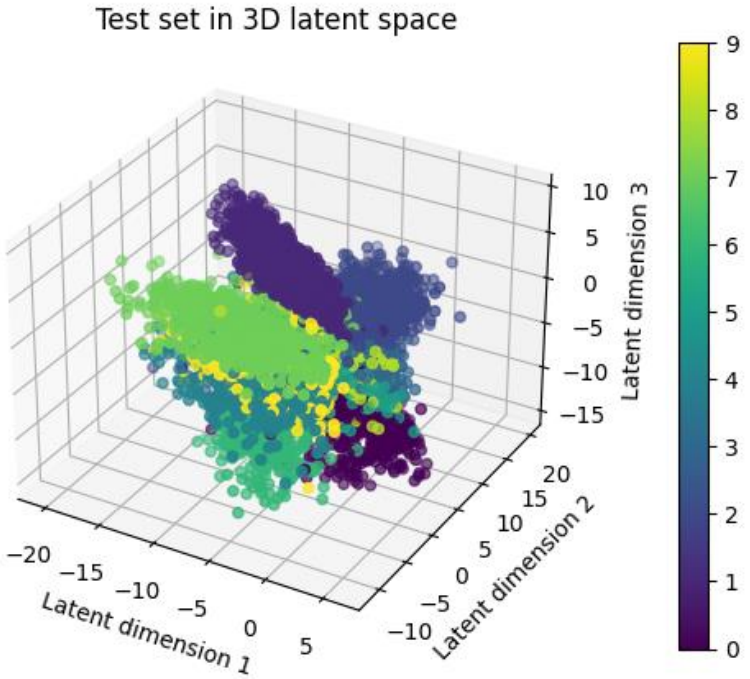
Autoencoder άσκησης

```
Epoch [1/20], Loss: 0.0408
Epoch [2/20], Loss: 0.0433
Epoch [3/20], Loss: 0.0375
Epoch [4/20], Loss: 0.0383
Epoch [5/20], Loss: 0.0352
Epoch [6/20], Loss: 0.0389
Epoch [7/20], Loss: 0.0324
Epoch [8/20], Loss: 0.0292
Epoch [9/20], Loss: 0.0316
Epoch [10/20], Loss: 0.0285
Epoch [11/20], Loss: 0.0337
Epoch [12/20], Loss: 0.0344
Epoch [13/20], Loss: 0.0344
Epoch [14/20], Loss: 0.0383
Epoch [15/20], Loss: 0.0347
Epoch [16/20], Loss: 0.0347
Epoch [17/20], Loss: 0.0285
Epoch [18/20], Loss: 0.0297
Epoch [19/20], Loss: 0.0357
Epoch [20/20], Loss: 0.0304
```

B) Στο latent space οι εικόνες έχουν πλέον τρία χαρακτηριστικά συνεπώς τα διαγράμματα που ακολουθούν έχουν 3 άξονες έναν για κάθε χαρακτηριστικό. Ακολουθεί το plot για το training set. Το colorbar στα δεξιά δείχνει απεικονίζει το πλήθος των κλάσεων με διαφορετικό χρώμα.



Συγκρίνοντας με το plot για το test set, είναι εμφανές πως η μορφή των δύο απεικονίσεων είναι παρόμοια με την διαφορά ότι το training set εμφανίζει πιο πυκνά τα σημεία λόγω μεγαλύτερης έκτασης που προκύπτει από τον διαχωρισμό του dataset. Συνεπώς, διατηρούνται τα χαρακτηριστικά της κατανομής των απεικονίσεων



Γ) Για το ερώτημα αυτό, αρχικά επιλέγουμε έναν αριθμό από εικόνες που θέλουμε να οπτικοποιήσουμε αφού πρώτα τις λάβουμε από το latent space και τις εισάγουμε στον decoder. Στην συνέχεια, γίνεται ένα reshape προκειμένου να έρθουν στην μορφή των εικόνων αυτού του dataset (28x28). Όπως φαίνεται και στην εικόνα που ακολουθεί, δεν είναι όλα τα δείγματα όμοια με τα πραγματικά. Πιο συγκεκριμένα, μπορούμε να διακρίνουμε τους αριθμούς 2,3,5,9 ενώ οι υπόλοιπες απεικονίσεις έχουν ακαθόριστη μορφή. Αυτό συμβαίνει αφού έχουμε μικρύνει τόσο πολύ το latent space που μεγάλο μέρος της πληροφορίας χάνεται. Όμως, από εκτέλεση σε εκτέλεση λόγω της τυχαιότητας επιλογής σημείων στο latent space τα αποτελέσματα ενδέχεται να διαφέρουν (οι αριθμοί που είναι ευδιάκριτοι). Τέλος, αξίζει να σημειωθεί πως δεδομένα που βρίσκονται σε κοντινή απόσταση στο latent space δεν σημαίνει ότι είναι στην πραγματικότητα κοντά κάτι που οφείλεται στην ελάττωση διαστάσεων.

Άσκηση 3:

Τα δέντρα αποφάσεων είναι μια μέθοδος επιβλεπόμενης μάθησης που χρησιμοποιείται για προβλήματα classification καθώς και regression. Ο αλγόριθμος αυτός δημιουργεί μια δομή όπως ένα δέντρο που αποτελείται από πολλαπλούς κόμβους αποφάσεων και έχεις ως «φύλλα» τις διάφορες επιλογές. Κάθε κόμβος στο δέντρο αυτό αντιπροσωπεύει ένα χαρακτηριστικό και η κατεύθυνση των ακμών του δέντρου δείχνει την ροή των αποφάσεων. Σε κάθε βήμα, ο αλγόριθμος διαλέγει το καλύτερο χαρακτηριστικό έτσι ώστε να διαχωρίσει τα δεδομένα σε ομάδες που ομοιάζουν περισσότερο. Για να γίνει ένα prediction, τα δεδομένα διασχίζουν το δέντρο από το root έως τα leafs ακολουθώντας τις αποφάσεις που γίνονται σε κάθε κόμβο. Ο κόμβος στον οποίο θα καταλήξουν αποτελεί καθοριστικό ρόλο για την πρόβλεψη.

Α) Όπως και στην άσκηση 1, χωρίζουμε το dataset σε train και test set με την συνάρτηση train_test_split. Ρυθμίζουμε την παράμετρο test_size = 0.3 ώστε το 30% του dataset να χρησιμοποιηθεί για testing.

Β) Αρχικά δημιουργούμε στα δεδομένα του training missing values σε ένα ποσοστό 10%. Στην συνέχεια δημιουργούμε ένα δέντρο απόφασης με την συνάρτηση DecisionTreeClassifier

δίνοντας ως όρισμα max_depth = 5 για να έχουμε βάθος 5 επίπεδα. Για τον υπολογισμό την ακρίβειας χρησιμοποιούμε την συνάρτηση accuracy_score αφού πρώτα με την συνάρτηση predict βρούμε τα labels που επιστρέφει ο ταξινομητής στο test set. Ο αντίστοιχος κώδικας επιστρέφει:

```
Prediction accuracy for test set: 0.9239766081871345
```

Καθώς στην υπόλοιπη άσκηση γίνεται αναφορά στα random forests, ας δώσουμε μια σύντομη περιγραφή για τον τρόπο λειτουργίας τους. Τα random forests είναι ένα σύνολο από δέντρα αποφάσεων (όπως είδαμε παραπάνω) και συνδυάζουν τις δυνατότητες των δέντρων με την τυχαιότητα ώστε να βελτιωθεί και να γενικευτεί το μοντέλο. Πιο συγκεκριμένα, κάθε δέντρο του random forest εκπαιδεύεται σε ένα διαφορετικό τυχαίο σετ δεδομένων με επανατοποθέτηση δειγμάτων ή χωρίς (bootstrapped samples) και χρησιμοποιεί ένα τυχαίο υποσύνολο χαρακτηριστικών ώστε να υπολογίσει την έξοδο. Στην συνέχεια, συνδυάζονται οι έξοδοι και λαμβάνεται η τελική απόφαση. Στα προβλήματα classification, για την τελική πρόβλεψη αξιοποιείται το majority vote μεταξύ των δέντρων. Γενικά, τα random forests εμφανίζουν μειωμένο κίνδυνο εμφάνισης overfitting και είναι πιο ανθεκτικά σε δεδομένα που περιέχουν θόρυβο.

Γ) Δεδομένου ότι η βιβλιοθήκη sklearn δεν μπορεί να διαχειριστεί τα δεδομένα NaN, για την δημιουργία random forest θα εκπαιδεύσουμε δέντρα για διαφορετικά features σε όλο το training set. Πιο συγκεκριμένα, θα δημιουργηθούν 100 ταξινομητές (δέντρα) όπου θα έχουν βάθος 3 και θα χρησιμοποιούν τυχαία 5 χαρακτηριστικά. Ο κώδικας για αυτό το ερώτημα δεν διαφέρει πολύ από αυτόν του προηγούμενου καθώς επαναλαμβάνει την εκπαίδευση και το testing του δέντρου για όσες φορές απαιτούνται για να σχηματιστεί το random forest. Για την επιλογή της τελικής κλάσης εφαρμόζεται majority voting για ποσοστό μεγαλύτερο του 50% (0.5). Η ακρίβεια που πετυχαίνει το RF είναι μεγαλύτερη από αυτή του απλού δέντρου και συγκεκριμένα:

```
Prediction accuracy for test set: 0.9649122807017544
```

Δ) Για την αποφυγή επανάληψης μεγάλου μέρους του κώδικα ο υπολογισμός της σημαντικότητας των χαρακτηριστικών για τον δεντρικό ταξινομητή και το random forest έγινε στα ερωτήματα β και γ αντίστοιχα. Για την εύρεση λοιπόν της σημαντικότητας χρησιμοποιήθηκε η συνάρτηση feature_importances_ . Η ιδιαιτερότητα που προκύπτει είναι κατά την εκτέλεση στο random forest κάποιες τιμές importance ήταν NaN. Για την αποφυγή αυτού του φαινομένου, κατά τον υπολογισμό του μέσου όρου της σημαντικότητας ενός χαρακτηριστικού δεν λήφθηκαν υπόψη οι μετρήσεις των ταξινομητών (δέντρων) που δίναν NaN. Έπειτα έγινε μια κανονικοποίηση ώστε τα importances να δίνουν ως άθροισμα 1. Ακολουθεί η έξοδος του κώδικα για την εκάστοτε περίπτωση.

Decision Tree

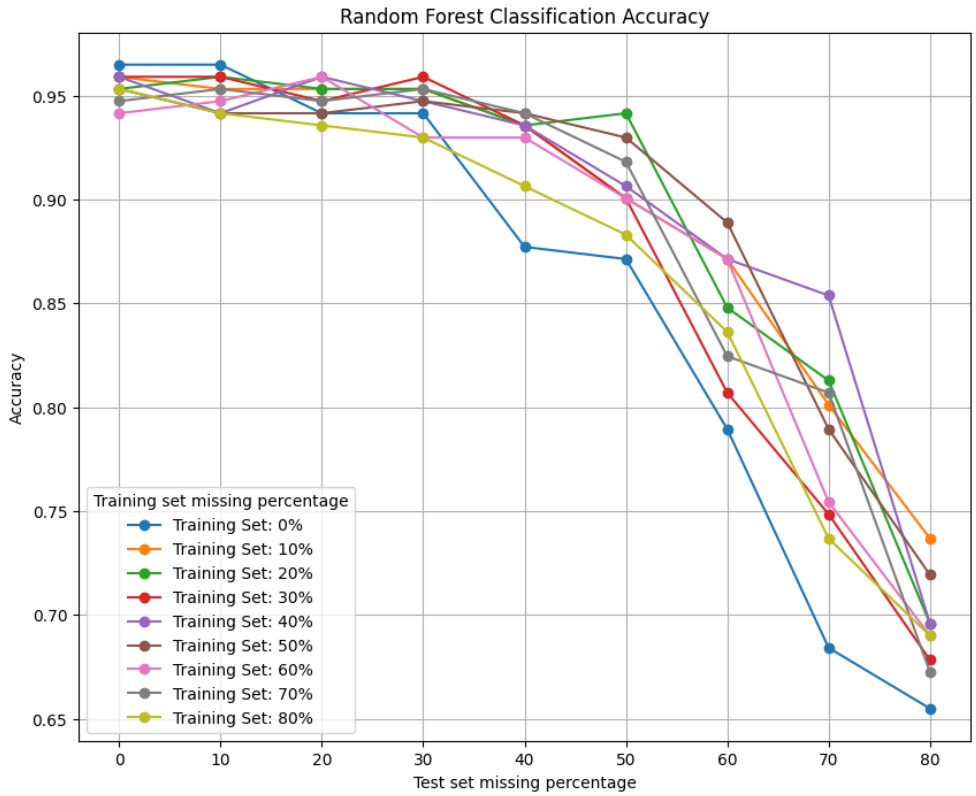
```
Feature 1: Importance = 0.0000
Feature 2: Importance = 0.0403
Feature 3: Importance = 0.0306
Feature 4: Importance = 0.0000
Feature 5: Importance = 0.0000
Feature 6: Importance = 0.0000
Feature 7: Importance = 0.0214
Feature 8: Importance = 0.0000
Feature 9: Importance = 0.0000
Feature 10: Importance = 0.0000
Feature 11: Importance = 0.0143
Feature 12: Importance = 0.0000
Feature 13: Importance = 0.0000
Feature 14: Importance = 0.0106
Feature 15: Importance = 0.0000
Feature 16: Importance = 0.0204
Feature 17: Importance = 0.0000
Feature 18: Importance = 0.0000
Feature 19: Importance = 0.0000
Feature 20: Importance = 0.0000
Feature 21: Importance = 0.0000
Feature 22: Importance = 0.0011
Feature 23: Importance = 0.0153
Feature 24: Importance = 0.2030
Feature 25: Importance = 0.0109
Feature 26: Importance = 0.0039
Feature 27: Importance = 0.0000
Feature 28: Importance = 0.6282
Feature 29: Importance = 0.0000
Feature 30: Importance = 0.0000
```

Random Forest

```
Feature 1: Importance = 0.0415
Feature 2: Importance = 0.0121
Feature 3: Importance = 0.0446
Feature 4: Importance = 0.0699
Feature 5: Importance = 0.0051
Feature 6: Importance = 0.0125
Feature 7: Importance = 0.0665
Feature 8: Importance = 0.0720
Feature 9: Importance = 0.0033
Feature 10: Importance = 0.0026
Feature 11: Importance = 0.0246
Feature 12: Importance = 0.0006
Feature 13: Importance = 0.0263
Feature 14: Importance = 0.0284
Feature 15: Importance = 0.0039
Feature 16: Importance = 0.0021
Feature 17: Importance = 0.0014
Feature 18: Importance = 0.0049
Feature 19: Importance = 0.0012
Feature 20: Importance = 0.0036
Feature 21: Importance = 0.0594
Feature 22: Importance = 0.0188
Feature 23: Importance = 0.1288
Feature 24: Importance = 0.0911
Feature 25: Importance = 0.0135
Feature 26: Importance = 0.0284
Feature 27: Importance = 0.0513
Feature 28: Importance = 0.1692
Feature 29: Importance = 0.0091
Feature 30: Importance = 0.0034
```

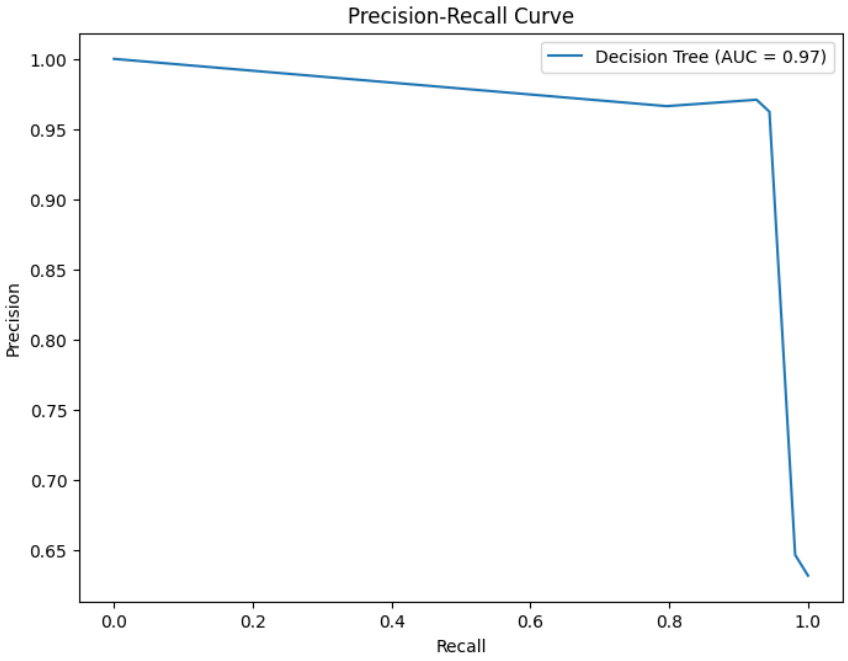
Παρατηρούμε αρχικά πως σε ένα τυχαίο forest κάθε χαρακτηριστικό έχει έστω και μια μικρή τιμή σημαντικότητας σε αντίθεση με το δέντρο όπου εκεί κάποια χαρακτηριστικά έχουν μηδενική σημαντικότητα και άλλα πολύ υψηλή. Αυτό συμβαίνει καθώς κάθε δέντρο του forest έχει διαφορετικά χαρακτηριστικά. Αξίζει να σημειωθεί πως το random forest σημείωσε μεγαλύτερη ακρίβεια στο test set. Ο λόγος που συμβαίνει αυτό είναι διότι το μοντέλο είναι πιο γενικευμένο λόγω της χρήσης πολλών επιμέρους δέντρων που δουλεύουν σε ξεχωριστά χαρακτηριστικά και με την χρήση του majority vote επιλέγουμε την κλάση που εμφανίζει πλειοψηφία.

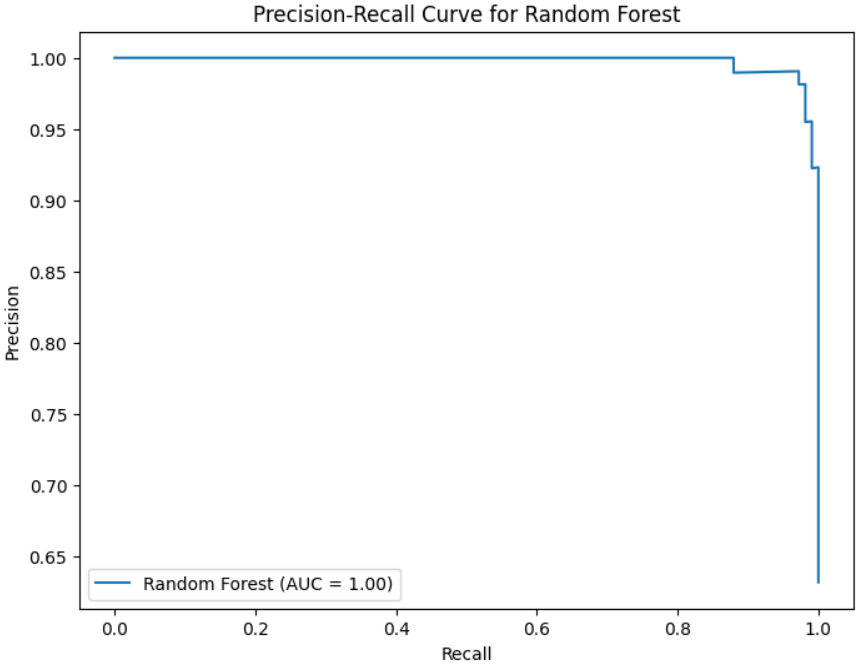
E) Για το ερώτημα αυτό, χρησιμοποιούμε τον κώδικα του ερωτήματος γ βάζοντας τον να τρέχει για τα μεταβλητά missing values των training και test set όπως ζητούνται. Το αποτέλεσμα φαίνεται στο παρακάτω διάγραμμα.



Παρατηρούμε πως όταν το training set δεν έχει missing values για διαρκώς αυξανόμενο ποσοστό missing values στο test set η απόδοση κατεβαίνει απότομα. Αντίστοιχα για πολύ μεγάλο αριθμό από NaN δεδομένα στο training set , το μοντέλο και πάλι ρίχνει το accuracy για μεγάλο ποσοστό απώλειας στο test set. Η βέλτιστη συμπεριφορά στο accuracy γίνεται αντιληπτή για ενδιάμεσες τιμές από missing data στο training set καθώς εκεί το μοντέλο φαίνεται να ανταποκρίνεται καλύτερα στο test set. Η χρήση του training set με όλα τα δεδομένα γενικά οδηγεί το μοντέλο σε overfitting πράγμα που δεν επιθυμητό. Αντίθετα, η μεγάλη απώλεια δεδομένων κατά την εκπαίδευση οδηγεί το μοντέλο να μάθει σε δεδομένα που ενδεχομένως δεν περιέχουν σημαντικά πρότυπα καθώς αυτά λείπουν. Τα random forest φαίνεται να έχουν καλή απόδοση και σε υψηλά ποσοστά χαμένων δεδομένων στο train set. Παρόλα αυτά, υπάρχει ευαισθησία στην ποικιλία των missing data στο test set διότι εάν τα patterns των τιμών που λείπουν διαφέρουν σημαντικά από αυτά της εκπαίδευσης η απόδοση θα μειωθεί.

ΣΤ) Από τα διαγράμματα που ακολουθούν παρατηρούμε πως το random forest έχει καλύτερη συμπεριφορά καθώς η καμπύλη precision-recall ομοιάζει με αυτή ενός ιδανικού ταξινομητή. Γενικά, η επιλογή υψηλής ευαισθησίας προτιμάται όταν η καταγραφή όλων των πραγματικών θετικών περιπτώσεων είναι κρίσιμη και οι συνέπειες των λανθασμένων αρνητικών προβλέψεων είναι υψηλές. Από την άλλη μεριά, η επιλογή υψηλής ακρίβειας προτιμάται όταν οι λανθασμένες θετικές προβλέψεις έχουν σοβαρές συνέπειες και η ακρίβεια είναι κρίσιμη. Δεδομένου ότι το πρόβλημα αφορά την πρόβλεψη κακοήθειας, είναι προτιμότερο να γίνει διάγνωση ενός ατόμου ότι είναι θετικό ενώ στην πραγματικότητα δεν είναι διότι έχει λιγότερες επιβλαβείς συνέπειες σε αντίθεση με το να προβλεφθεί αρνητικό και να είναι θετικό. Κατά συνέπεια μας ενδιαφέρει να υπάρχει υψηλή ευαισθησία.


Precision-Recall Curve
Decision Tree (AUC = 0.97)
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve for Random Forest
Random Forest (AUC = 1.00)
Precision
Recall
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.0
0.2
0.4
0.6
0.8
1.0