Neural Networks – Machine Learning

«Τεχνητή Νοημοσύνη»

Εργασία 2

Ονοματεπώνυμο: Τριανταφυλλίδης Παναγιώτης **ΑΜ:** 3200199

Ονοματεπώνυμο: Δημήτριος Πατάπιος Παραράς **ΑΜ:** 3200148

Ονοματεπώνυμο: Χρήστος Πολύζος **ΑΜ:** 3190173

Αρχιτεκτονική(Δομή) – Τρόπος Χρήσης

Σημείωση: Δεν περιγράφονται οι αλγόριθμοι μηχανικής μάθησης που έχουμε αναπτύξει εδώ αλλά σε μεταγενέστερο σημείο. Εδώ αναφέρονται για την δομή της αρχιτεκτονικής

- Fetch Data: Φορτώνουμε με την βοήθεια του TensorFlow API τα κείμενα που θα χρησιμοποιήσουμε ως παραδείγματα εκπαίδευσης & αξιολόγησης στους αλγορίθμους μηχανικής μάθησης.
- Create Vocabulary: Στην συνέχεια δημιουργούμε το λεξικό των κειμένων, ανάλογα με τις υπερ-παραμέτρους που θα περάσει ο χρήστης (m, n, k).
- Create binary vectors: Έπειτα εκτελούμε τον ανάλογο κώδικα για να μετατρέψουμε κάθε κείμενο σε μία λίστα από αναπαράσταση 0, 1, όπου κάθε θέση της λίστας δηλώνει και μια θέση(λέξη) στο λεξικό και το ανάλογο 0 ή 1 δηλώνει αν υπάρχει ή όχι η λέξη στο κείμενο. Εδώ καλούμε την create_vocabulary, με ορίσματα τα m, n, k.
- Information Gain & Entropy: Συνάρτηση που μας επιτράπηκε να την δανειστούμε από το φροντιστήριο 8 και για ένα πίνακα που περιέχει τις κλάσεις [0, 1], και μια ιδιότητα (λέξη) του κειμένου μας βρίσκει το κέρδος πληροφορίας.
    - Παράμετροι: class_(NumPy. array), feature(int)
    - Returns: ig-information_gain
- Naive Bayes, ID3, Random Forest & RNN (BigRu) είναι οι αλγόριθμοι και το νευρωνικό δίκτυο αντίστοιχα, τα οποία οφείλουμε να κάνουμε compile πριν τρέξουμε τον κώδικα για τον σχηματισμό των αντίστοιχων καμπυλών. Περιγράφονται πιο κάτω.
- Custom_curve: Για συγκεκριμένο αλγόριθμο ή νευρωνικό δίκτυο, αναλαμβάνει να κατασκευάσει σε ένα sub_plot 4 διαγράμματα (learning curve, precision curve , recall curve, f1 curve. Πριν την εκτέλεση χωρίζουμε τα παραδείγματα εκπαίδευσης με βάση την παράμετρο n_splits (e.g. n_splits = 5 χωρίζουμε τα παραδείγματα ανά 5.000) σε n_splits ομάδες παραδειγμάτων. Αρχικά εκπαιδεύουμε τον αλγόριθμο ή νευρωνικό δίκτυο για την $1^{η}$ ομάδα παραδειγμάτων σε και αφότου τον εκπαιδεύσουμε, καλούμε την συνάρτηση predict() του estimator, ώστε να προβλέψουμε τις κατηγορίες, τόσο για όλα τα κείμενα της ομάδας παραδειγμάτων που εκπαιδεύσαμε τον estimator , όσο και για **όλα** τα κείμενα αξιολόγησης. Κάθε ένα από τα δύο αποτέλεσμα του predict τα αποθηκεύουμε σε ξεχωριστές μεταβλητές, ώστε καλώντας τις συναρτήσεις

accuracy_score, precision_score, recall_score, f1_score με τις κατάλληλες παραμέτρους, να αποθηκεύσουμε στις λίστες train_acc, test_acc, train_prec, test_prec, train_rec, test_rec, train_f, test_f τα αποτελέσματα των παραπάνω συναρτήσεων. Τέλος, συγχωνεύουμε την 1η ομάδα παραδειγμάτων εκπαίδευσης με την 2η ομάδα (συνεπώς έχουμε 10.000 αν θυμηθούμε το παράδειγμα που δώσαμε παραπάνω) και ακολουθούμε την ίδια διαδικασία για την νέα (συγχωνευμένη) ομάδα παραδειγμάτων εκπαίδευσης. Ακολουθούμε σταθερά τα ίδια βήματα μέχρι να φθάσουμε να έχουμε εκτελέσει τον αλγόριθμο για την τελευταία (συγχωνευμένη) ομάδα παραδειγμάτων εκπαίδευσης (δηλαδή όλα τα 25.000 παραδείγματα εκπαίδευσης που είχαμε αναφέρει σαν παράδειγμα). Γενικά, ο σκοπός είναι να εκπαιδεύουμε τον αλγόριθμο μηχανικής μάθησης μια μικρή ομάδα παραδειγμάτων εκπαίδευσης και να τον ελέγχουμε σε όλα τα παραδείγματα αξιολόγησης. Στην συνέχεια αποθηκεύαμε τόσο το accuracy, precision, recall, f1 της ομάδας παραδειγμάτων εκπαίδευσης αλλά και όλων των παραδειγμάτων αξιολόγησης στις αναφερόμενες λίστες. Έπειτα προσθέτουμε παραδείγματα εκπαίδευσης στην ομάδα παραδειγμάτων εκπαίδευσης και συνεχίζουμε την διαδικασία. Έτσι κατασκευάζουμε τις καμπύλες accuracy, precision, recall, f1 curve με τα στοιχεία των λιστών και τυπώνει τις ανάλογες καμπύλες.

  - Παράμετροι: **estimator**(neural network ή machine learning algorithm), **x_train**(train examples= numpy.array ή text of strings), **y_train**(list with elements 0 or 1), **x_test**(test examples= numpy.array ή text of strings), **y_test**(list with elements 0 or 1), **n_splits**(int), **title**(string), **zoom_out**(bool).
  - Returns: **data** (dictionary= λεξικό με τις λίστες για την δημιουργία πινάκων του estimator)

- compare_two_classification_algorithms: Η ίδια εκτέλεση με τον custom_curve αλλά τώρα ακολουθεί την ίδια διαδικασία για δύο αλγορίθμους μηχανικής μάθησης (όχι νευρωνικό δίκτυο). Αποτυπώνει πάνω στο ίδιο plot τις καμπύλες των δύο αλγορίθμων για σύγκριση.
  - Παράμετροι: **estimator1**(machine learning algorithm),**estimator2**(machine learning algorithm), **x_train**(**train_examples**= numpy.array ή text of strings), **y_train**(list with elements 0 or 1), **x_test**(test examples= numpy.array ή text of strings), **y_test**(list with elements 0 or 1), **n_splits**(int), **title**(string), **zoom_out**(bool), **title1**(string), **title2**(string).
  - Returns:
    - data1(dictionary= λεξικό με τις λίστες για την δημιουργία πινάκων του estimator1)
    - data2(dictionary= λεξικό με τις λίστες για την δημιουργία πινάκων του estimator2)

- loss plot: Αναλαμβάνει την κατασκευή των καμπυλών μεταβολής σφάλματος, συναρτήσει του αριθμού των εποχών του RNN BigRu.
    - Παράμετροι:
        - His (History.object)
        - Loss (string= 'loss')
- Pandas_tables: Κατασκευάζει με βάση το λεξικό τους πίνακες του αλγορίθμου μηχανικής μάθησης ή του νευρωνικού δικτύου.
    - Παράμετροι:
        - X_train_b (NumPy.array): Παραδείγματα εκπαίδευσης
        - Data (dictionary)
        - Splits(int)

Μέθοδοι Τεχνητής Νοημοσύνης

Υλοποιήσαμε το Naïve Bayes, ID3 & Random Forest. Κάθε μέλος της ομάδας ανέλαβε την υλοποίηση και την περιγραφή της επιλογής του.Επίσης στο τέλος περιγράφουμε το RNN BigRu

Naive Bayes:

- **__init__(self, alpha= 1):** Ορίζουμε δύο λίστες, τις:
    - x1c1array: αποθηκεύονται οι δεσμευμένες πιθανότητες p(x(i)=1 | c=1). Εύκολα υπολογίζεται p(x(i)=1 | c=1) = 1 - p(x(i)=0 | c=1).
    - x1c0array: αποθηκεύονται οι δεσμευμένες πιθανότητες p(x(i)=1 | c=0). Εύκολα υπολογίζεται p(x(i)=1 | c=1) = 1 - p(x(i)=0 | c=1).
    - alpha: εκτιμήτρια Laplace.
- **fit (self, x_train_binary, y_train):** Με βάση τα κείμενα εκπαίδευσης, γεμίζουμε τους πίνακες που ορίσαμε στον constructor την κλάσης. Σε κάθε υπολογισμό χρησιμοποιούμε και τον Laplace estimator.
- **Predict (self, x_train_binary):** Κάνει τους απαραίτητους υπολογισμούς με βάση τα δεδομένα σε κάθε κείμενο ελέγχου και βρίσκει ποια από τις δύο πιθανότητες είναι μεγαλύτερη max(p(c=0|X), p(c=1|X)).

ID3:

- **ID3_Tree:** κλάση κόμβου/υπόδενδρου
  - **__init__(self):**
    - Tag: η τιμή της λέξης με βάση την οποία σπάει ο κόμβος σε δύο υποδένδρα. Είναι 0 ή 1
    - Feature: Η λέξη που σπάει ο κόμβος σε δύο υποδένδρα. Είναι ο αριθμός της στήλης.
    - child_nodes: μια λίστα με τα δύο υπόδενδρα του κόμβου.
    - Decision: Η απόφαση/κατηγορία που καταλήγει σε αυτόν τον κόμβο. Αν είναι 1 ή 0 ο κόμβος είναι φύλλο
  - **Is_leaf(self):** ελέγχει αν ο κόμβος είναι φύλο με κριτήριο το decision.
  - **_create_child:** προσθέτει ένα παιδί στον κόμβο.
- **ID3:** υλοποίηση βασικής κλάσης
  - **__init__ (self, labels, max_depth = 10, dc = 0):**
    - Labels: Το σύνολο των χαρακτηριστικών/λέξεων. Κάθε λέξη είναι ο δείκτης μιας στήλης
    - Max_depth: Το συνολικό βάθος του δένδρου
    - Dc: προ-επιλεγμένη απόφαση
    - Tree: Το δένδρο που προκύπτει μετά από κάθε εκτέλεση της μεθόδου fit.
  - **Most_IG (self, x_train_i, y_train_i, labels):** Βρίσκει εκείνο το χαρακτηριστικό/λέξη που προσφέρει το μεγαλύτερο κέρδος πληροφορίας.
  - **Fitting_tree (self, x_train_b, y_train_b, labels, dc, depth):** Εδώ ο κώδικας είναι γραμμένος με βάση τον ψευδοκώδικα των διαφανειών. Ουσιαστικά χωρίζεται σε δύο περιπτώσεις:
    - Βάση Αναδρομής:
      - Αν δεν υπάρχουν διαθέσιμα παραδείγματα, επέστρεψε κόμβο με προεπιλεγμένη επιλογή σαν απόφαση.
      - Αν όλα τα παραδείγματα είναι μόνο μια κατηγορία, τότε επέστρεψε κόμβο με την συγκεκριμένη κατηγορία σαν απόφαση.

- Αν δεν υπάρχει κανένα χαρακτηριστικό τότε επέστρεψε κόμβο με απόφαση να είναι η πλειοψηφική από τα εναπομείναντα παραδείγματα.
- Αν φθάσει το μέγιστο βάθος, τότε επέστρεψε τον κόμβο με απόφαση την πλειοψηφική κατηγορία.
- Αν τα θετικά έχουν ξεπεράσει το 95%, τότε επέστρεψε κόμβο με απόφαση 1.Αντίστοιχα, αν τα αρνητικά έχουν ξεπεράσει το 95%, τότε επέστρεψε κόμβο με απόφαση 0(Pruning Cases).

▪ Αναδρομική κλήση: Βρες το καλύτερο χαρακτηριστικό, εκείνο με το μεγαλύτερο κέρδος πληροφορίας .Για κάθε κατηγορία που το συναντάμε στα παραδείγματα εκπαίδευσης της αναδρομής:

- Κράτησε εκείνα τα παραδείγματα που έχουν την συγκεκριμένη κατηγορία
- Αφαίρεσε από την λίστα των λέξεων , την λέξη/χαρακτηριστικό με το κέρδος πληροφορίας.
- Κάλεσε αναδρομικά την συνάρτηση με τις ανανεωμένες παραμέτρους.

- **Fit (self, x_train_b, y_train_b):** Καλεί την fitting_tree και ορίζει το δένδρο που είναι το αποτέλεσμα εκπαίδευσης.
- **Predict_sample (self, x_sample, tree):** Διασχίζει το δένδρο με τα χαρακτηριστικά/λέξεις του κειμένου μέχρι να φθάσει σε φύλλο και να επιστρέψει την απόφαση του φύλλου.
- **Predict (self, x_train_binary):** Χρησιμοποιώντας την predict_sample, για κάθε παράδειγμα βρίσκει την κατηγορία που πρέπει να ανήκει. Επιστρέφει όλα τα αποτελέσματα σε μία λίστα/έναν πίνακα.

Random Forest:

- **__init__ (self, num_trees, max_depth):**
  - **M:** ο αριθμός των χαρακτηριστικών που θα λάβουμε υπόψιν
  - **Num_trees:** ο αριθμός των δένδρων που θα φτιάξει ο αλγόριθμος
  - **Max_depth:** το βάθος των δένδρων
  - **Ls_trees:** λίστα αποθήκευσης δένδρων
- **Subsample (self, x_data, y_data):** Ανακατεύει τα παραδείγματα και επιστρέφει έναν νέο πίνακα παραδειγμάτων, με πιθανά διπλότυπα.

- **Subfeature (self, x_data):** Ανακατεύει τα χαρακτηριστικά και επιστρέφει έναν πίνακα ανακατεμένων χαρακτηριστικών.
- **Fit (self, x_train_b, y_train_b):** κατασκευάζει num_trees δένδρα σε πλήθος με τυχαία παραδείγματα και χαρακτηριστικά/λέξεις κάθε φορά.
- **Predict (self, x_test_b):** Προβλέπει για κάθε κείμενο την κατηγορία που ανήκει, με πλειοψηφία από τις αποφάσεις των δένδρων που έχουμε αποθηκεύσει στο ls_trees με βάση την πλειοψηφία των επιμέρους δένδρων.

RNN – BigRu

- **__init__ (self, vocabulary, VOCAB_SIZE, SEQ_MAX_LENGTH, epochs, verbose, batch_size):** Ορίζουμε το λεξικό του vectorizer μέσω της συνάρτησης create_vec και αρχικοποιούμε τις μεταβλητές epochs, verbose και batch_size της κλάσης.
- **create_vec (self, VOCAB_SIZE, SEQ_MAX_LENGTH, vocabulary):** Ορίζουμε το vectorizer με λεξικό που έχουμε ήδη δημιουργήσει σύμφωνα με τα n, m, k (μεταβλητές εκφώνησης) ώστε να έχουμε κοινό μέτρο σύγκρισης για τη σύγκριση του rnn με τους αλγορίθμους μηχανικής μάθησης.
- **get_bigru (self, num_layers, emb_size, h_size):** Ορίζουμε το rnn με τη σιγμοειδή συνάρτηση ενεργοποίησης.
- **fit (self, x_train_b, y_train_b):** Αρχικοποιούμε το rnn μέσω της συνάρτησης get_bigru με loss function το BinaryCrossEntropy, κάθε φορά που εκπαιδεύουμε το μοντέλο μας. Έπειτα καλούμε τη συνάρτηση fit της κλάσης model.
- **predict (self, x_test_b):** Καλούμε τη predict της κλάσης model και τα αποτελέσματα της τα στρογγυλοποιούμε σε 0 και 1 μέσω της συνάρτησης NumPy.round().
- **get_history (self, x_train_b, y_train_b, validation_split):** Επιστρέφουμε το history object καθώς εκπαιδεύουμε το rnn σε παραδείγματα εκπαίδευσης και ανάπτυξης (με βάση το ποσοστό του validation_split).

Μέρος Α':

Περιγραφή: Ορίζουμε ένα λεξικό παραλείποντας τις κ = 1000 σπανιότερες λέξεις και τις n = 50 συχνότερες λέξεις. Τέλος κρατάμε τις m = 3000 συχνότερες λέξεις. Έχουμε 22.500 κείμενα εκπαίδευσης(training), 2500 κείμενα ανάπτυξης(validation) και 25000 κείμενα αξιολόγησης(testing). Τα validation κείμενα χρησιμοποιήθηκαν για να οριστούν οι υπερπαράμετροι όπως το max_depth στον ID3, το πλήθος δέντρων και ο αριθμός των χαρακτηριστικών – λέξεων στο Random_Forest

Learning, precision, recall, f1 for Naive Bayes

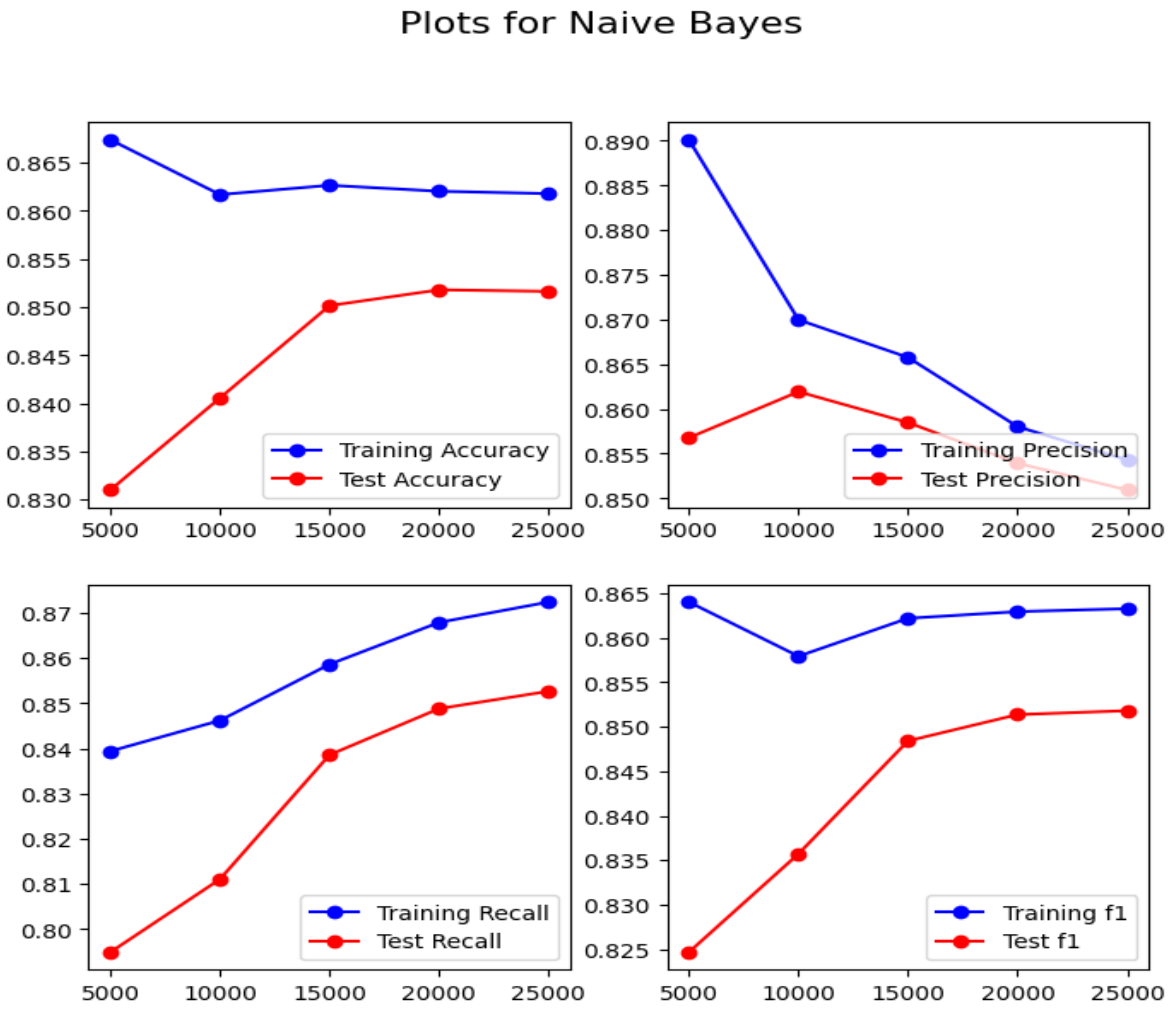
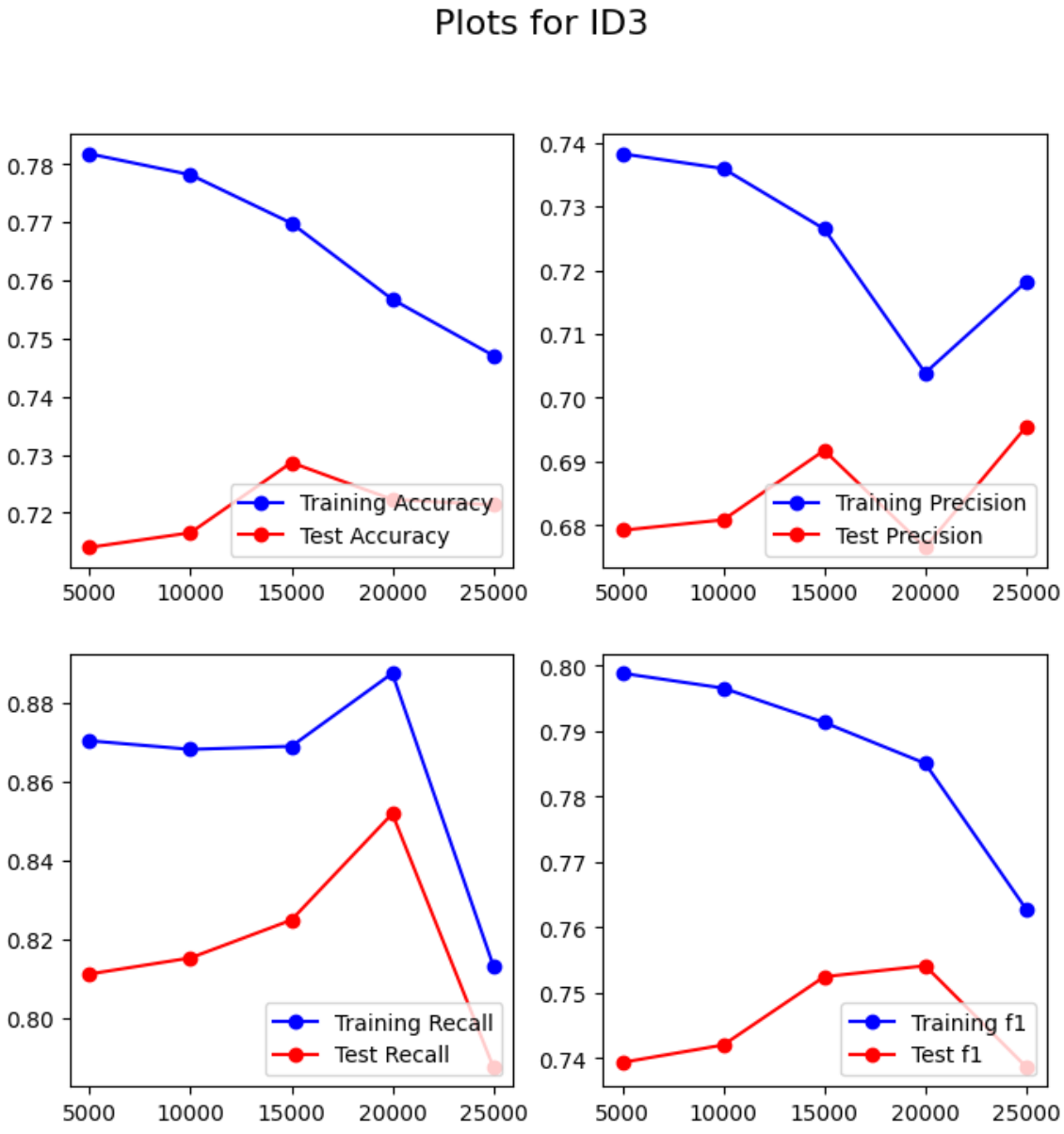


Table data

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.867400	0.83100	0.890110	0.856773	0.839378	0.79488	0.864000	0.824667
0.861700	0.84056	0.869973	0.861928	0.846169	0.81104	0.857906	0.835710
0.862667	0.85016	0.865762	0.858477	0.858609	0.83856	0.862170	0.848401
0.862050	0.85180	0.858018	0.853924	0.867879	0.84880	0.862920	0.851354
0.861800	0.85164	0.854289	0.850938	0.872400	0.85264	0.863250	0.851788

Learning, precision, recall, f1 for ID3

Παρατήρηση: Βρήκαμε πως το βέλτιστο βάθος για το δέντρο του ID3 από τις τιμές 1 έως 10 είναι το μέγιστο βάθος των 10 (συγκεκριμένα επιλέξαμε τις 10 αυτές τιμές διότι ο αλγόριθμος παίρνει πολύ χρόνο στην εκτέλεση). Συγκεκριμένα για κάθε βάθος εκπαιδεύαμε τον αλγόριθμο στα κείμενα εκπαίδευσης και τον αξιολογούσαμε στα δεδομένα ανάπτυξης. Επιλέξαμε το βάθος με το μεγαλύτερο accuracy score για τα παρακάτω διαγράμματα.



Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.781800	0.71404	0.738241	0.679239	0.870229	0.81112	0.798820	0.739344
0.778200	0.71656	0.735932	0.680877	0.868053	0.81520	0.796551	0.742008
0.769867	0.72864	0.726465	0.691761	0.868826	0.82480	0.791294	0.752445
0.756750	0.72228	0.703906	0.676558	0.887335	0.85176	0.785048	0.754117
0.747080	0.72136	0.718191	0.695465	0.813280	0.78760	0.762784	0.738670

Learning, precision, recall, f1 for Random Forest

Παρατήρηση: Για τις υπερ-παραμέτρους m (number of features) και το πλήθος των δένδρων, έχουμε δοκιμάσει να συγκρίνουμε μικρές random τιμές (καθώς ο αλγόριθμος είναι πολύ χρονοβόρος), και καταλήξαμε πως προτιμότερη τιμή για το πλήθος δένδρων είναι το 3 (βέβαια το 5 είναι λίγο καλύτερη τιμή, αλλά πολύ χρονοβόρο και σε εκτέλεση έπαιρνε 2 ώρες για πολλές ιδιότητες), αλλά και από ιδιότητες δοκιμάσαμε ανάμεσα στις 100, 200, 300, 500, και βρήκαμε το 500 ως το βέλτιστο. Οι συγκρίσεις έγινα με 22.500 δεδομένα αξιολόγησης σε 2.500 δεδομένα ανάπτυξης.

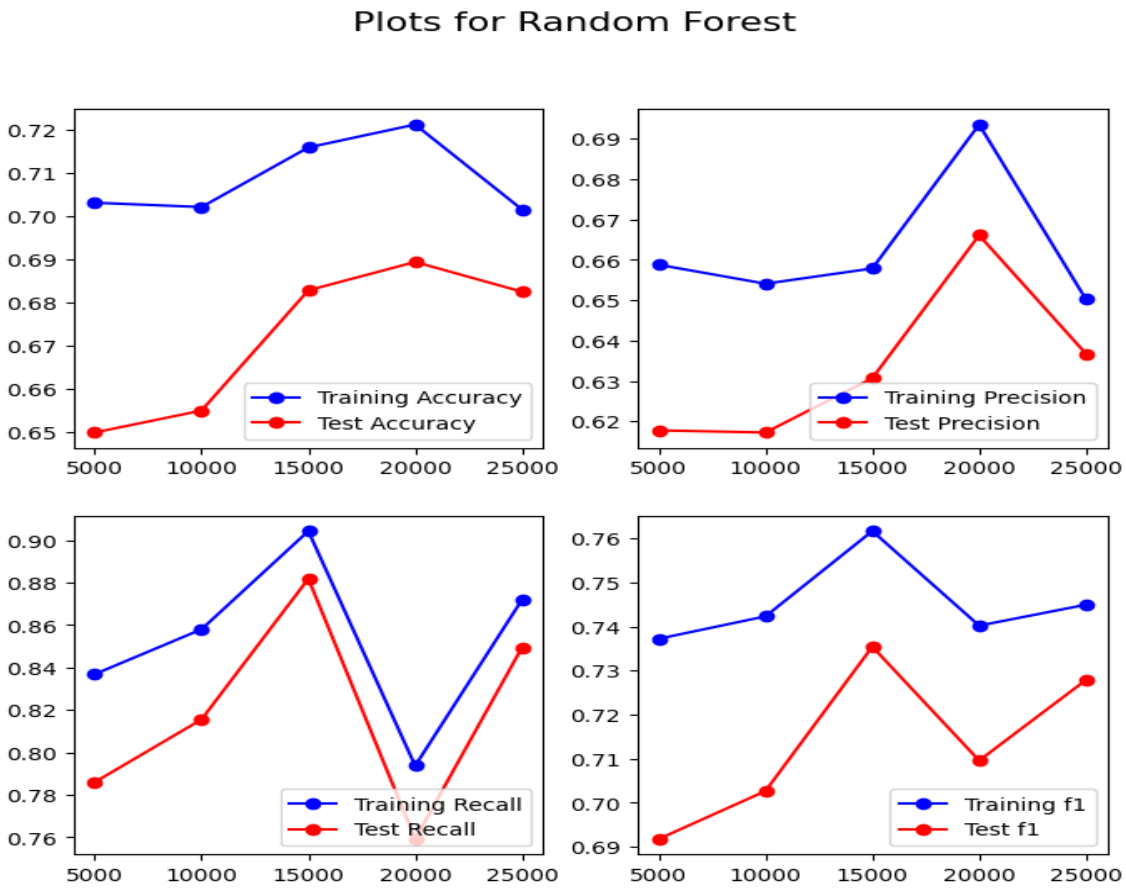
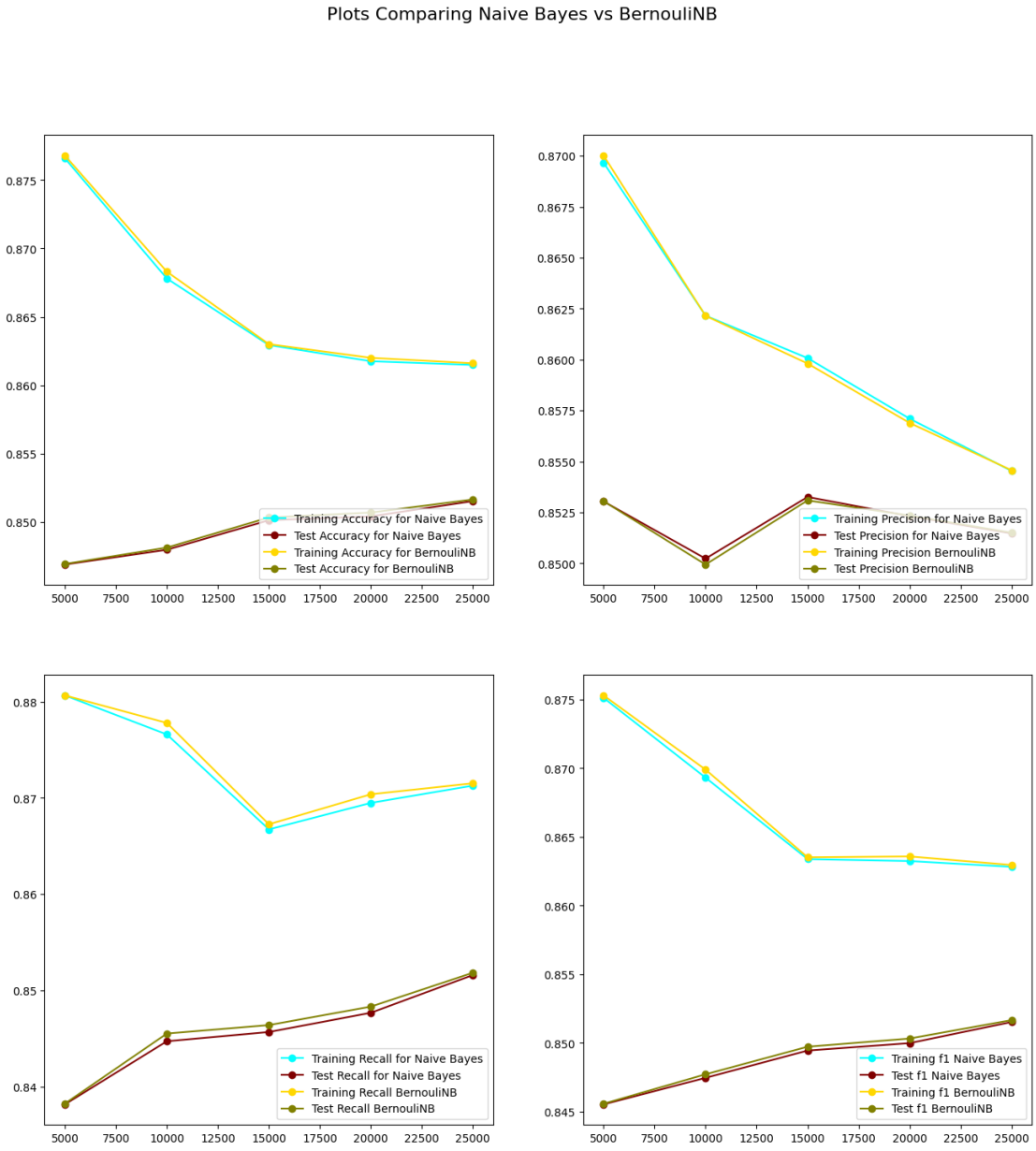


Table Data

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.70300	0.64980	0.658760	0.617745	0.836882	0.78592	0.737215	0.691758
0.70200	0.65488	0.654069	0.617220	0.858057	0.81552	0.742304	0.702647
0.71580	0.68272	0.657846	0.630664	0.904408	0.88192	0.761670	0.735424
0.72110	0.68928	0.693489	0.666082	0.793648	0.75912	0.740196	0.709564
0.70144	0.68240	0.650203	0.636723	0.872000	0.84944	0.744943	0.727859

Μέρος Β':

Naïve Bayes vs BernoulliNB



Naïve Bayes data

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.876600	0.84688	0.869670	0.853037	0.880652	0.83816	0.875126	0.845533
0.867800	0.84796	0.862157	0.850229	0.876595	0.84472	0.869316	0.847466
0.862933	0.85012	0.860074	0.853257	0.866729	0.84568	0.863389	0.849452
0.861750	0.85040	0.857101	0.852317	0.869483	0.84768	0.863247	0.849992
0.861480	0.85152	0.854531	0.851464	0.871280	0.85160	0.862824	0.851532

Bernoulli NB data

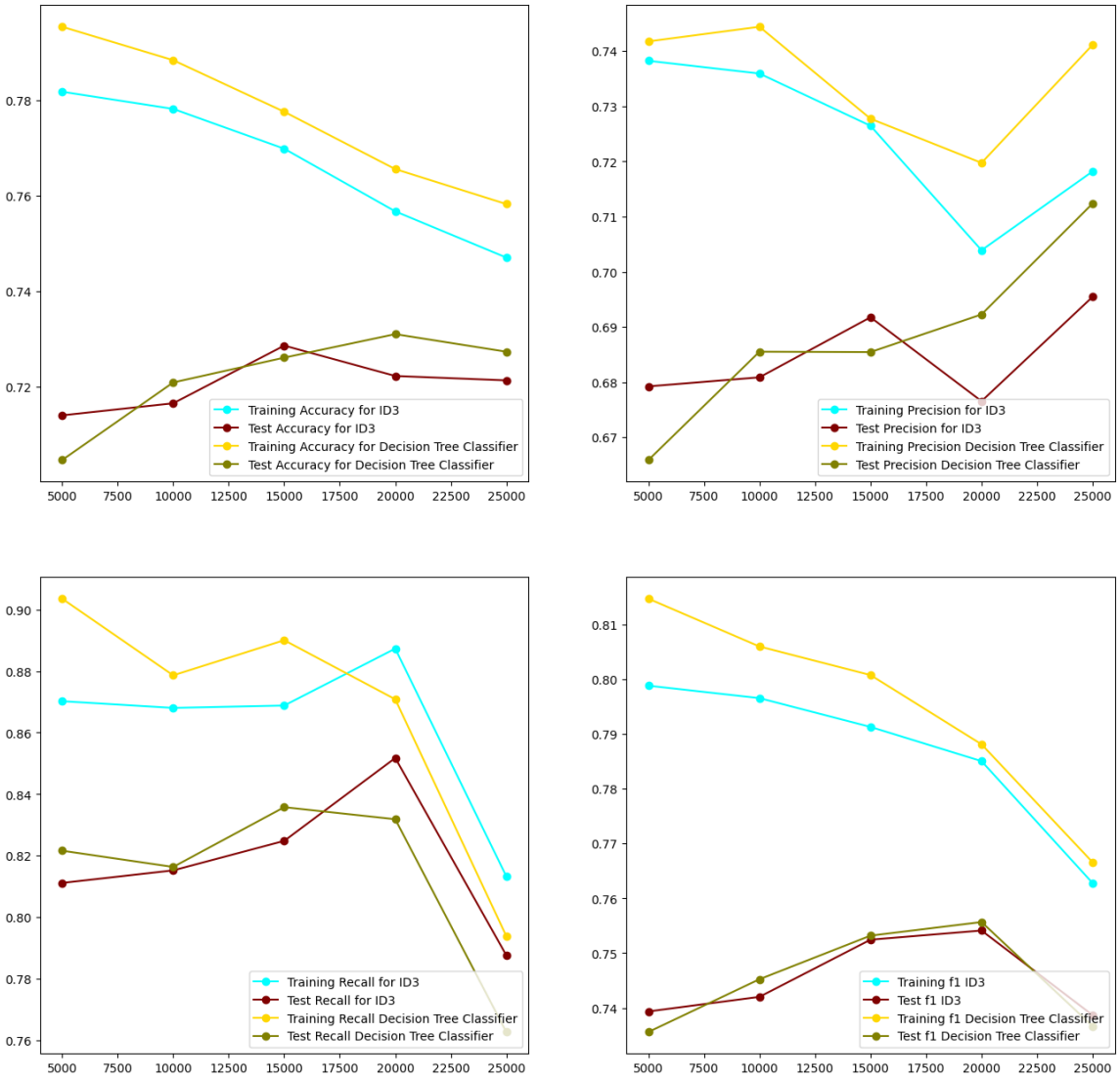
Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.8768	0.84692	0.870020	0.853049	0.880652	0.83824	0.875304	0.845580
0.8683	0.84812	0.862150	0.849940	0.877791	0.84552	0.869900	0.847724
0.8630	0.85032	0.859807	0.853088	0.867263	0.84640	0.863519	0.849731
0.8620	0.85068	0.856891	0.852343	0.870380	0.84832	0.863582	0.850327
0.8616	0.85164	0.854565	0.851499	0.871520	0.85184	0.862959	0.851670

Παρατήρηση:

Είναι εύκολο να γίνει αντιληπτό, πως τόσο από τα διαγράμματα, όσο και από τους πίνακες, οι αλγόριθμοι βγάζουν κοινά αποτελέσματα με μικρές αποκλίσεις. Για αυτό προτιμήθηκε εξάλλου και η ζουμαρισμένη έκδοση ώστε να φανεί καλύτερα.

ID3 vs Decision Tree Classifier

Plots Comparing ID3 vs Decision Tree Classifier



ID3 data

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.781800	0.71404	0.738241	0.679239	0.870229	0.81112	0.798820	0.739344
0.778200	0.71656	0.735932	0.680877	0.868053	0.81520	0.796551	0.742008
0.769867	0.72864	0.726465	0.691761	0.868826	0.82480	0.791294	0.752445
0.756750	0.72228	0.703906	0.676558	0.887335	0.85176	0.785048	0.754117
0.747080	0.72136	0.718191	0.695465	0.813280	0.78760	0.762784	0.738670

Decision Tree Classifier

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.79540	0.70468	0.741755	0.665889	0.903576	0.82160	0.814707	0.735594
0.78840	0.72092	0.744411	0.685522	0.878649	0.81632	0.805978	0.745225
0.77760	0.72612	0.727746	0.685454	0.890069	0.83576	0.800764	0.753181
0.76560	0.73104	0.719746	0.692277	0.870855	0.83184	0.788123	0.755669
0.75828	0.72736	0.741131	0.712375	0.793840	0.76264	0.766580	0.736651

Παρατηρήσεις:
Accuracy

Ο αλγόριθμος της βιβλιοθήκης είναι καλύτερος προς την ακρίβεια τόσο στα δεδομένα ανάπτυξης, όσο και στα δεδομένα αξιολόγησης, σταθερά κατά 0.5% - 0,8% από τον δικό μας. Μόνο σε μερικά σημεία παρατηρούμε το αντίθετο.

Precision

Παρομοίως για το precision, υπάρχουν μικρές αποκλίσεις στις τιμές, με τον decision tree classifier να παρουσιάζει καλύτερη απόδοση από τον id3. Βέβαια, σε μερικά σημεία συμβαίνει το αντίθετο.

Recall

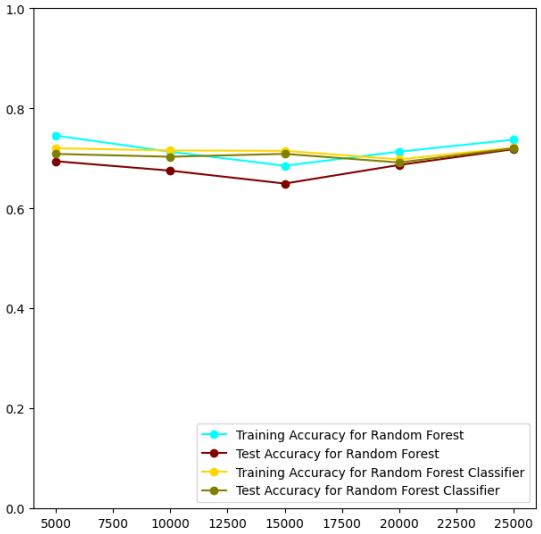
Εδώ παρατηρούμε πως ο αλγόριθμος μας είναι καλύτερος από του ski kit learn. Είναι αναμενόμενο αφού το precision του decision tree classifier ήταν μεγαλύτερο από του id3 όπως αναφέρθηκε, άρα παρατηρήθηκε το trade – off σε αυτήν την περίπτωση.

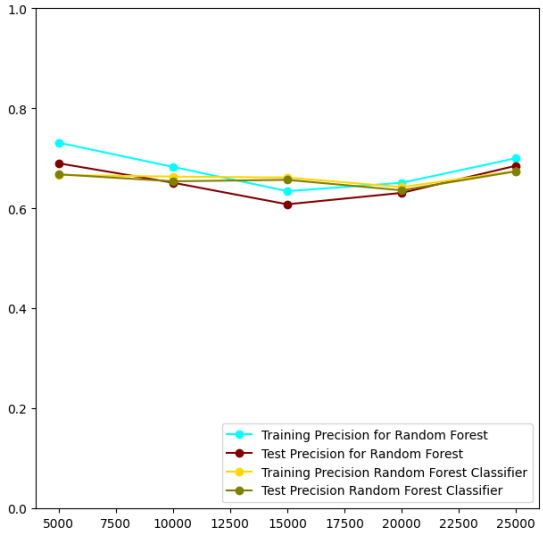
F1

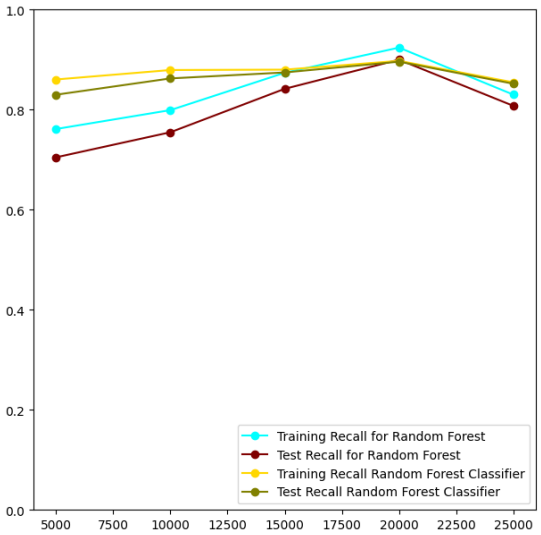
Σε αυτήν την περίπτωση βλέπουμε ότι όσο έχουμε μεγαλύτερο training data, τόσο πλησιάζουν οι τιμές και των δύο αλγορίθμων κοντά. Μπορούμε να παρατηρήσουμε και την σύγκλιση των καμπυλών για αυτό.

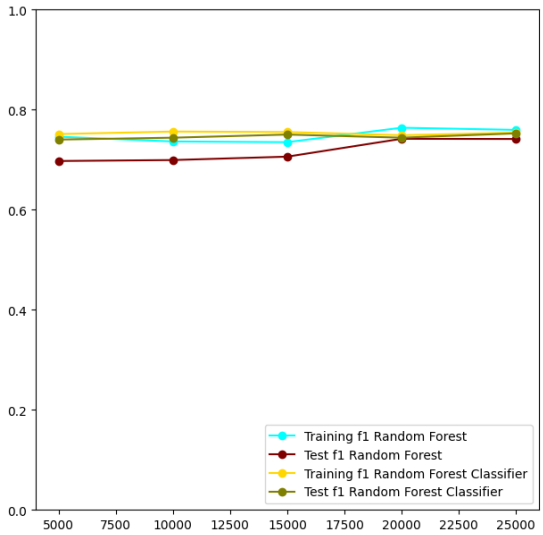
Random Forest vs Random Forest Classifier

Plots Comparing Random Forest vs Random Forest Classifier









Random Forest

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.74540	0.69412	0.731221	0.690090	0.761303	0.70472	0.745959	0.697328
0.71310	0.67536	0.682910	0.651319	0.799043	0.75480	0.736426	0.699251
0.68520	0.64960	0.634347	0.608092	0.873666	0.84160	0.735017	0.706040
0.71330	0.68668	0.651035	0.630882	0.923981	0.89984	0.763858	0.741732
0.73728	0.71824	0.700162	0.685150	0.830000	0.80760	0.759572	0.741353

Random Forest Classifier

Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.72020	0.70876	0.666667	0.668062	0.860285	0.82984	0.751200	0.740215
0.71570	0.70316	0.663457	0.654087	0.879187	0.86240	0.756238	0.743936
0.71500	0.70892	0.661518	0.656988	0.879936	0.87432	0.755253	0.750232
0.69795	0.69168	0.642511	0.636105	0.897479	0.89584	0.748888	0.743954
0.72116	0.71992	0.674560	0.674009	0.854640	0.85184	0.753997	0.752562

Παρατηρήσεις:

Accuracy

Βλέπουμε πως το δικός μας αλγόριθμος παρουσιάζει καλύτερη ακρίβεια στα δεδομένα εκπαίδευσης από ότι ο αλγόριθμος της βιβλιοθήκης ski kit-learn, εκτός από το σημείο που αφορά τα 15.000 παραδείγματα. Στα δεδομένα αξιολόγησης βέβαια, υπερτερεί ο αλγόριθμος της βιβλιοθήκης. Οι διαφορές είναι πάρα πολύ μικρές, κάτω από το 1% ορισμένες φορές.

Precision

Παρατηρούμε ότι ο αλγόριθμος της ski kit learn έχει μικρότερο precision, εκτός από ένα σημείο του δικού μας στα δεδομένα εκπαίδευσης. Στα δεδομένα αξιολόγησης αντίθετα συμβαίνουν πολλές διακυμάνσεις , όπου στο τέλος στα 25.000, ο δικός μας έχει μεγαλύτερο precision.

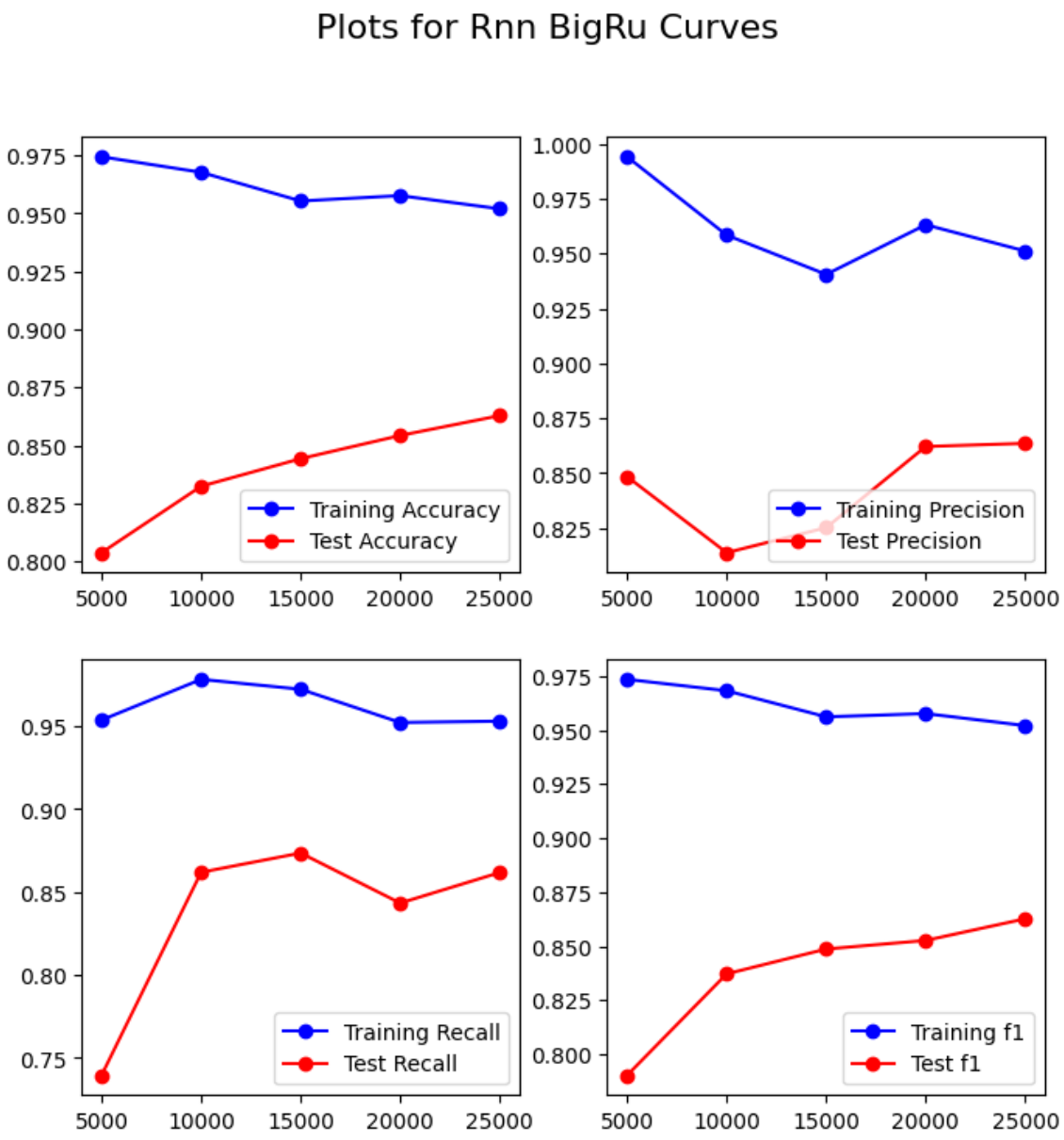
Recall

Το ακριβώς αντίθετο με ότι περιγράψαμε στο precision συμβαίνει εδώ. Αυτό είναι φυσιολογικό καθώς παρατηρούμε το trade-off του precision & recall στις προτιμήσεις.

F1

Βλέπουμε ότι το f1 score στα δεδομένα αξιολόγησης στον αλγόριθμό μας μεταβάλλεται και αυξάνεται απότομα , αλλά δε ξεπερνά το f1_score στα δεδομένα αξιολόγησης του αλγορίθμου του ski kit learn. Αντίθετα, στα δεδομένα εκπαίδευσης, f1_score είναι μεγαλύτερο στον δικό μας αλγόριθμο από ότι εκείνου της βιβλιοθήκης.

Μέρος Γ':



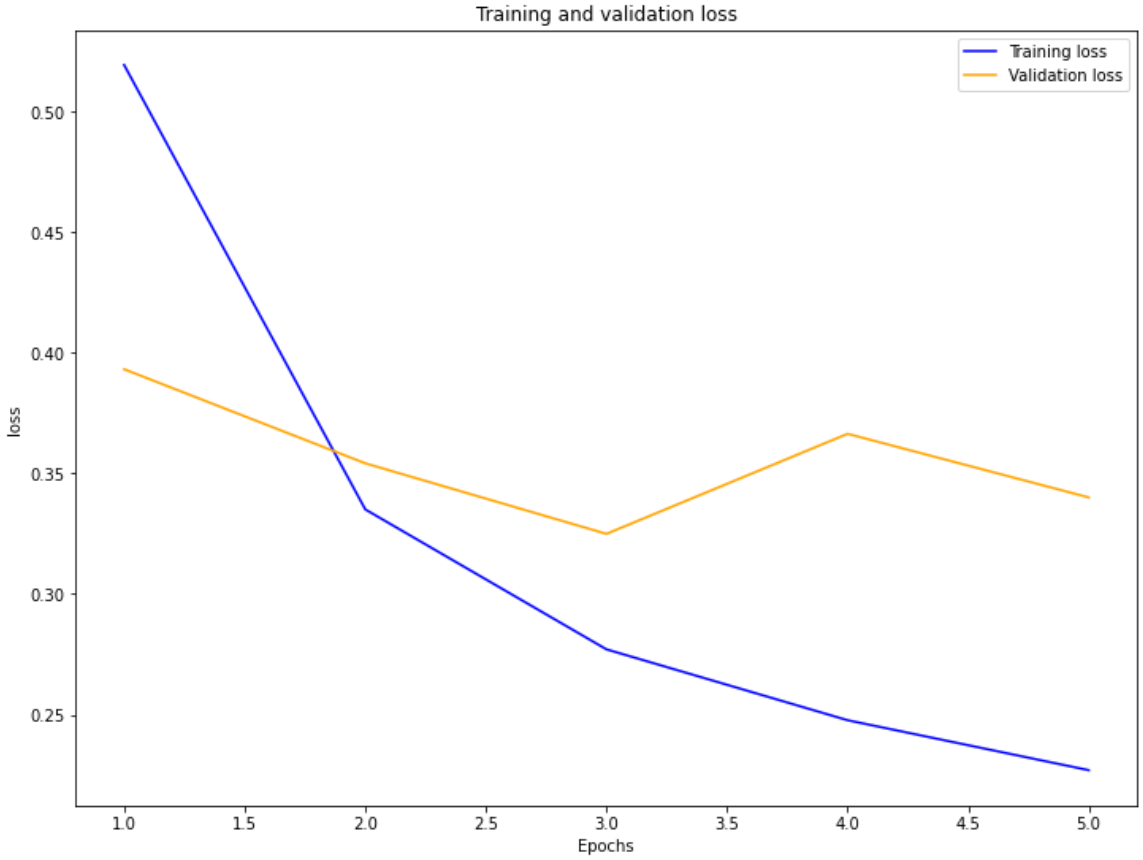
Train Accuracy	Test Accuracy	Train Precision	Test Precision	Train Recall	Test Recall	Train F1	Test F1
0.974400	0.80372	0.994055	0.848463	0.953564	0.73952	0.973389	0.790254
0.967700	0.83236	0.958390	0.813855	0.978070	0.86184	0.968130	0.837161
0.955333	0.84424	0.940501	0.825246	0.972118	0.87344	0.956048	0.848659
0.957650	0.85424	0.963112	0.862179	0.952077	0.84328	0.957563	0.852625
0.951960	0.86280	0.951130	0.863614	0.952880	0.86168	0.952004	0.862646

Παρατήρηση:

Συγκριτικά με τους υπόλοιπους αλγορίθμους, τόσο τους δικούς μας, όσο και της βιβλιοθήκης ski kit learn, το RNN διαθέτει καλύτερα αποτελέσματα σε όλες τις μετρικές μονάδες. Αυτό συμβαίνει καθώς ο αριθμός των εποχών τον βοηθάει να μαθαίνει καλύτερα κατά την εκπαίδευση και να αποδίδει κατά την αξιολόγηση.

Καμπύλες στην μεταβολή σφάλματος στα δεδομένα εκπαίδευσης & ανάπτυξης

- Χρησιμοποιήσαμε την συνάρτηση loss_plot με validation_split = 0.2.



Με το διάγραμμα παρατηρούμε πως με το πέρασμα των εποχών, διακρίνεται μια σταδιακή μείωση τόσο του validation loss, όσο και του training loss, φυσιολογικό, προσαρμόζεται σε νέα δεδομένα με τα βάρη που έχει δημιουργήσει, και συνεπώς έχουμε μεγαλύτερη ακρίβεια.