ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ 2 ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

Όνομα	**Αριθμός Μητρώου**	**e-mail**
Περγαντής Νικόλαος	3210163	p3210163@aueb.gr
Φωτογιαννόπουλος Δημήτριος	3210214	p3210214@aueb.gr
Τουμαζάτος Δημήτριος	3210199	p3210199@aueb.gr

Πριν την επεξήγηση των αλγορίθμων μηχανικής μάθησης, θα θέλαμε να αναφέρουμε πως στα αρχεία makeTestData και makeTrainData γίνεται η μετατροπή των reviews σε διανύσματα με 1 και 0 με βάση το λεξιλόγιο που επιλέξαμε (most_common_words) και υλοποιήσαμε μέσα στο makeTrainData, όπου και επιλέξαμε να πάρουμε τις 1700 πιο συχνές λέξεις αφαιρώντας τις 130 συχνότερες. Τέλος θέλουμε να ενημερώσουμε πως η εκτέλεση και το train των αλγορίθμων γίνεται εντός του myExecutionFile καθώς και η εμφάνιση των μετρήσεών τους.

ΜΕΡΟΣ Α

1) Naïve Bayes

Δημιουργήσαμε μια κλάση NaiveBayes η οποία αποτελείται από 4 μεθόδους. Φυσικά αρχικά μία __init__() για αρχικοποιήσεις. Στην συνέχεια ακολουθεί μία πολύ βασική συνάρτηση, η fit() εντός της οποίας υπολογίζεται η πιθανότητα για κάθε feature από τα trainData να ανήκει στα θετικά ή στα αρνητικά παραδείγματα. Πιο συγκεκριμένα, διαχωρίζουμε τα παραδείγματα εκπαίδευσης σε αρνητικά και θετικά, στην συνέχεια για κάθε feature υπολογίζουμε την πιθανότητα, μία για τα θετικά και μία για τα αρνητικά και στο τέλος υπολογίζουμε την συνολική πιθανότητα ένα παράδειγμα να είναι θετικό και την αντίστοιχη πιθανότητα να είναι αρνητικό. Τώρα στην calculateProb υπολογίζουμε την classification πιθανότητα έχοντας γνωστή πλέον την κατηγορία. Σε αυτό το σημείο να τονίσουμε πως χρησιμοποιούμε τον λογάριθμο ώστε να αποφύγουμε το σενάριο στο οποίο η πιθανότητα είναι πολύ μικρή και ο υπολογιστής την στρογγυλοποιήσει στον μηδέν. Αυτή λοιπόν η συνάρτηση χρησιμοποιείται εντός της predict() εντός της οποίας υπολογίζουμε την πιθανότητα ένα παράδειγμα να είναι θετικό ή αρνητικό με την χρήση της calculateProb και βασιζόμενοι πάντα στο διάνυσμά της. Ανάλογα με το ποια πιθανότητα είναι μεγαλύτερη η συνάρτηση δίνει την πρόβλεψή της, ενώ στην ακραία περίπτωση που οι πιθανότητες είναι ίσες δίνεται ως πρόβλεψη η κατηγορία που έχει γενικά μεγαλύτερη πιθανότητα με βάση τα trainData.

Ακολουθούν οι μετρήσεις και τα διαγράμματα σχετικά με την αποδοτικότητα του αλγορίθμου που υλοποιήσαμε:

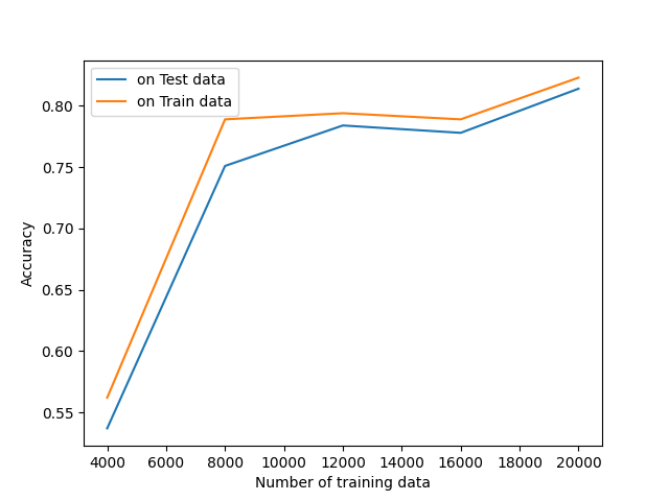
	Accuracy	
No. of Training Data	Train	Test
4000	0.562	0.537
8000	0.789	0.751
12000	0.794	0.784

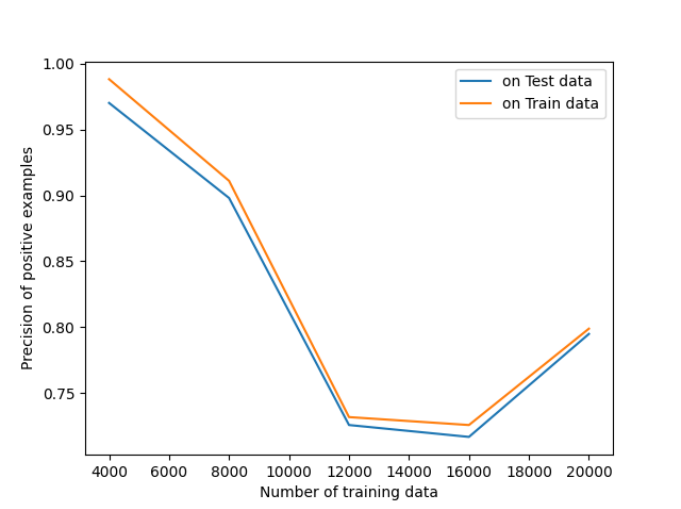
16000	0.789	0.778
20000	0.823	0.814

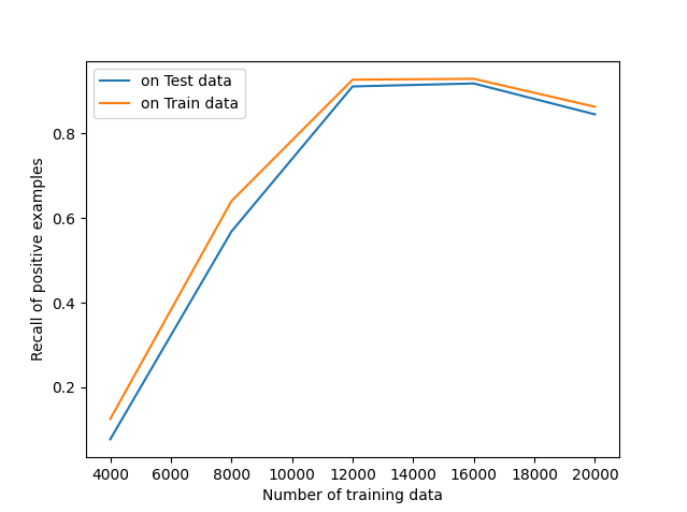
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.988	0.97
8000	0.911	0.898
12000	0.732	0.726
16000	0.726	0.717
20000	0.799	0.795

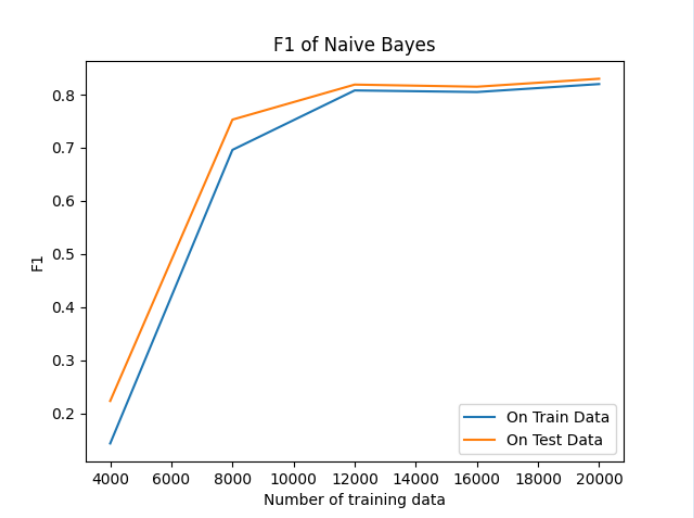
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.125	0.077
8000	0.640	0.568
12000	0.927	0.911
16000	0.929	0.918
20000	0.863	0.845

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.223	0.143
8000	0.753	0.696
12000	0.819	0.808
16000	0.815	0.805
20000	0.83	0.82









2) Random Forest

Πριν ξεκινήσουμε την ανάλυση του αλγορίθμου, αναφέρουμε πως για τον Random Forest χρησιμοποιήσαμε τον ID3 που δόθηκε στο φροντιστήριο με μικρές παρεμβάσεις που θεωρήσαμε αναγκαίες. Υλοποιήσαμε μια κλάση RandomForest η οποία φυσικά και αυτή έχει __init()__ για αρχικοποιήσεις. Υπάρχει μια συνάρτηση train() στην οποία γίνεται η δημιουργία των δέντρων (επιλέξαμε να φτιάξουμε 100) με την χρήση κάποιων τυχαίων features από τα train δεδομένα (επιλέξαμε να χρησιμοποιήσουμε 4). Κάθε φορά που δημιουργείται ένα νέο δέντρο το κάνουμε train με χρήση της fit() από τον ID3. Τώρα, στην συνάρτηση test() αθροίζουμε τις προβλέψεις τον δέντρων και στην περίπτωση που τα περισσότερα δέντρα κατέταξαν το παράδειγμα ως θετικό το κατατάσσουμε στην ανάλογη κατηγορία, αλλιώς το κατατάσσουμε στην αρνητική.

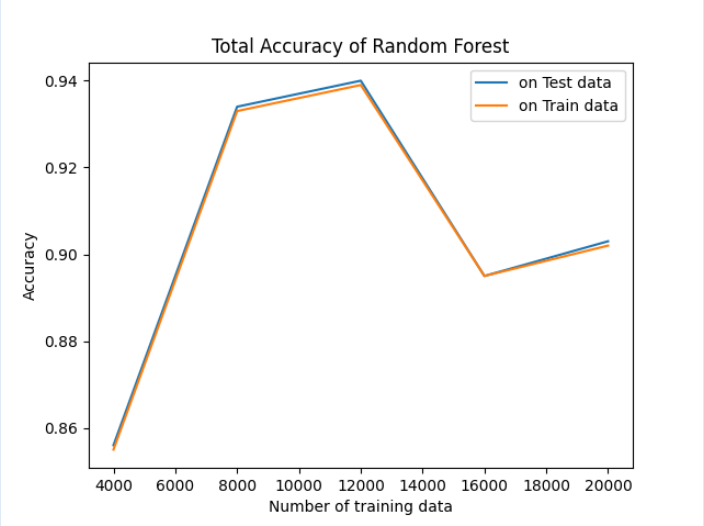
Ακολουθούν οι μετρήσεις και τα διαγράμματα σχετικά με την αποδοτικότητα του αλγορίθμου που υλοποιήσαμε:

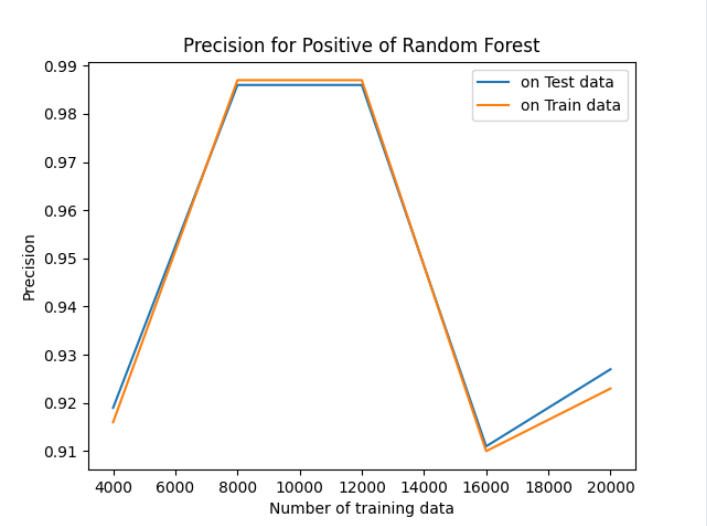
	Accuracy	
No. of Training Data	Train	Test
4000	0.855	0.856
8000	0.933	0.934
12000	0.939	0.94
16000	0.895	0.895
20000	0.902	0.903

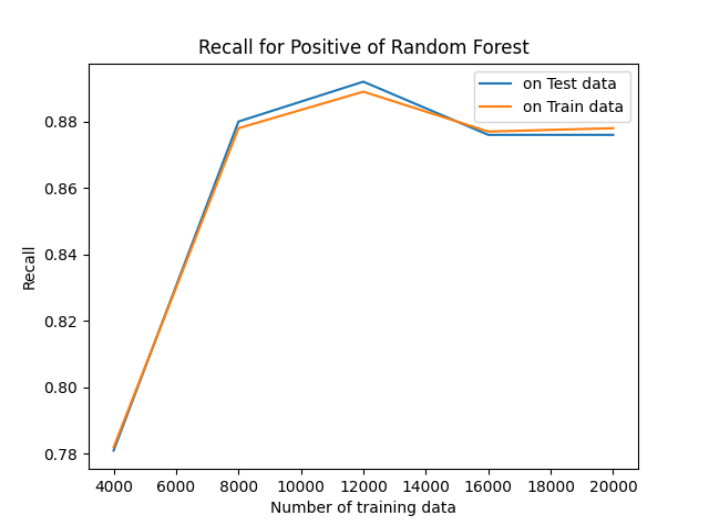
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.916	0.919
8000	0.987	0.986
12000	0.987	0.986
16000	0.91	0.911
20000	0.923	0.927

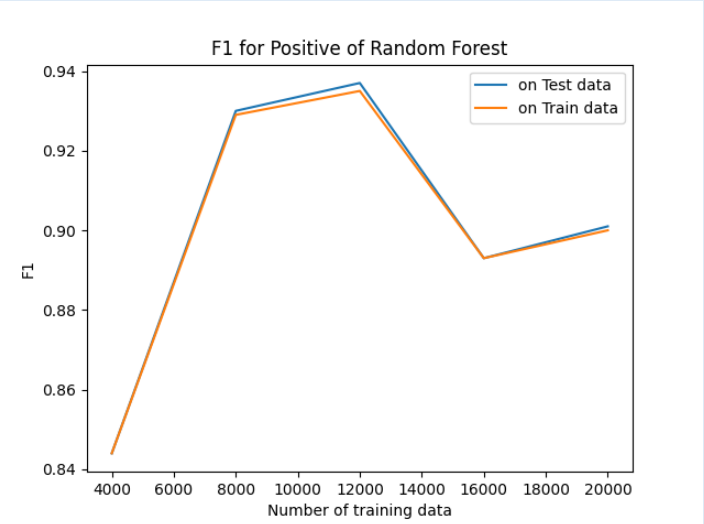
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.782	0.781
8000	0.878	0.88
12000	0.889	0.892
16000	0.877	0.876
20000	0.878	0.876

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.844	0.844
8000	0.929	0.93
12000	0.935	0.937
16000	0.893	0.893
20000	0.9	0.901









3) Ada Boost

Για τον Ada Boost έχουμε υλοποιήσει δύο κλάσεις, την κλάση AdaBoost αλλα και την κλάση SingleDepthTree που χρησιμοποιείται ως weak learner από την κλάση AdaBoost. Πιο συγκεκριμένα, αποφασίσαμε η κλάση AdaBoost να δημιουργεί 100 SingleDepthTrees. Αρχικά η κλάση SingleDepthTree έχει , όπως όλες, __init__() για αρχικοποιήσεις. Αρχικά, στην κλάση υπάρχει η μέθοδος giniIndex(), η οποία υπολογίζει την πιθανότητα ένα παράδειγμα να ανήκει στην κατηγορία 1 με δεδομένο πως το x είναι 0 και την πιθανότητα να ανήκει στην κατηγορία 1 με δεδομένο πως το x είναι 1. Από εκεί και πέρα, υπολογίζουμε τον μετρητή giniIndex και τον τοποθετούμε σε μια λίστα, από την οποία λίστα η μέθοδος επιστρέφει το feature με το μικρότερο giniIndex το οποίο είναι το καλύτερο feature για την δημιουργία δέντρου. Αυτή η μέθοδος χρησιμοποιείται στην fit() η οποία χρησιμοποιεί το feature που της έδωσε για να φτιάξει το δέντρο βάθους 1. Η fit() υπολογίζει τις πιθανότητες (C = 1 | X = 0), (C = 1 | X = 1), (C = 0 | X = 0), (C = 0 | X = 1) και ανάλογα με το ποια είναι μεγαλύτερη δημιουργεί το δέντρο και ορίζει κατηγορία. Επίσης, υπάρχουν οι μέθοδοι predict() που με βάση το επιλεγμένο feature δίνει πρόβλεψη για το παράδειγμα και η predict_row() που κάνει το ίδιο απλά αποφασίζει με βάση το επιλεγμένο feature της γραμμής.

Τώρα, εντός του Adaboost, εκτός της __init__, υπάρχει η fit() η οποία υλοποιεί τον αλγόριθμο του Adaboost, δηλαδή χρησιμοποιεί την predict() της SingleDepthTree για την πρόβλεψη του weak learner, στην συνέχεια υπολογίζει τα σφάλματα με βάση τα βάρη του κάθε παραδείγματος. Αν το συνολικό σφάλμα μίας υπόθεσης είναι μεγαλύτερο του 0.5 αφαιρείται το αντίστοιχο feature από την λίστα και η υπόθεση αυτή δεν χρησιμοποιείται στο τελικό μοντέλο. Στην συνέχεια ανάλογα με το αν μία υπόθεση τα πήγε καλά ή όχι αρχικοποιούμε το βάρος της και μειώνουμε τα βάρη των παραδειγμάτων που κατατάχθηκαν σωστά. Στο τέλος απλά χρησιμοποιούμε την μέθοδο normalizeWeights() που υλοποιήσαμε, η οποία απλά διαμορφώνει τα βάρη ώστε να αθροίζουν στο 1. Τέλος, υλοποιήσαμε και μία predict() για την κλάση AdaBoost η οποία χρησιμοποιεί την predict_row() και βγάζει αθροίσματα σχετικά με το πόσα δέντρα κατέταξαν το παράδειγμα στην μία κατηγορία και πόσα στην άλλη. Ανάλογα με την πλειοψηφία παίρνεται απόφαση, ενώ αν οι ψήφοι των δύο κατηγοριών ισοβαθμούν, επιλέγεται μία κατηγορία τυχαία.

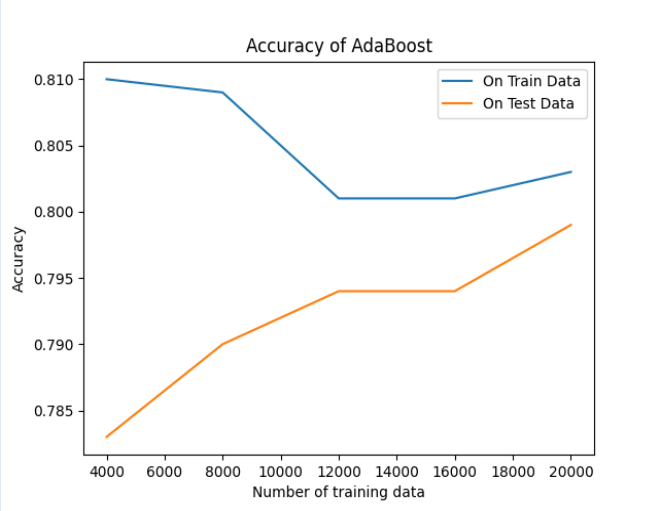
Ακολουθούν οι μετρήσεις και τα διαγράμματα σχετικά με την αποδοτικότητα του αλγορίθμου που υλοποιήσαμε:

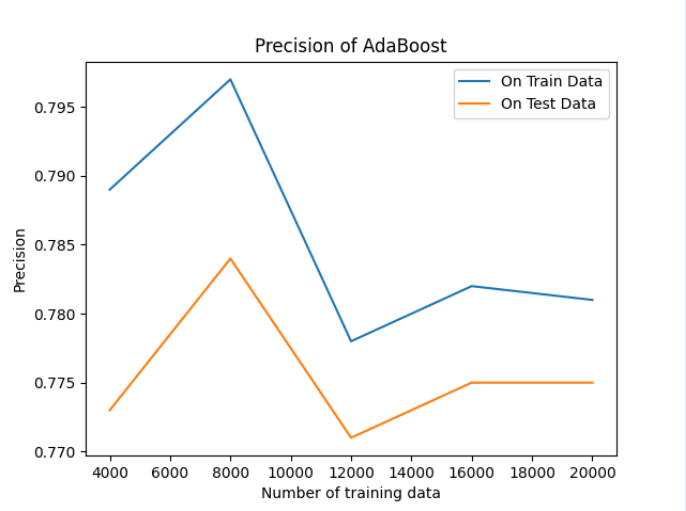
	Accuracy	
No. of Training Data	Train	Test
4000	0.81	0.783
8000	0.809	0.79
12000	0.801	0.794
16000	0.801	0.794
20000	0.803	0.799

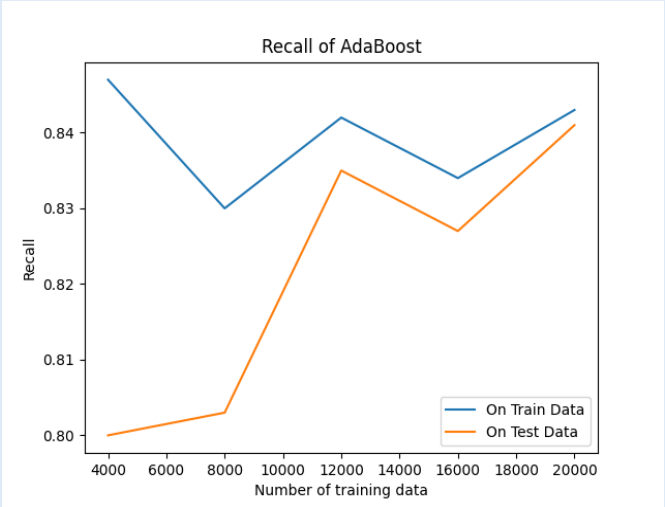
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.789	0.773
8000	0.797	0.784
12000	0.778	0.771
16000	0.782	0.775
20000	0.781	0.775

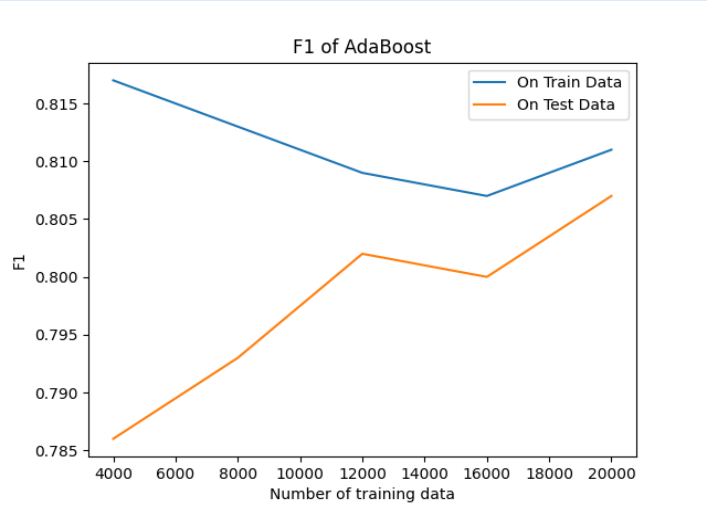
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.847	0.8
8000	0.83	0.803
12000	0.842	0.835
16000	0.834	0.827
20000	0.843	0.841

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.817	0.786
8000	0.813	0.793
12000	0.809	0.802
16000	0.807	0.8
20000	0.811	0.807


Accuracy of AdaBoost
On Train Data
On Test Data
Accuracy
Number of training data
0.810
0.805
0.800
0.795
0.790
0.785
4000 6000 8000 10000 12000 14000 16000 18000 20000


Precision of AdaBoost
On Train Data
On Test Data
Precision
Number of training data
0.795
0.790
0.785
0.780
0.775
0.770
4000 6000 8000 10000 12000 14000 16000 18000 20000


Recall of AdaBoost
On Train Data
On Test Data
Recall
Number of training data
0.84
0.83
0.82
0.81
0.80
4000 6000 8000 10000 12000 14000 16000 18000 20000


F1 of AdaBoost
On Train Data
On Test Data
F1
Number of training data
0.815
0.810
0.805
0.800
0.795
0.790
0.785
4000 6000 8000 10000 12000 14000 16000 18000 20000

ΜΕΡΟΣ Β

Οι υλοποιήσεις των αλγορίθμων του SciKitLearn βρίσκονται στο αρχείο algorithmsFromSciKitLearn.py όπου στην αρχή του γίνεται το διάβασμα των train και test data.

Σύγκριση Naïve Bayes (Our implementation vs SciKit)

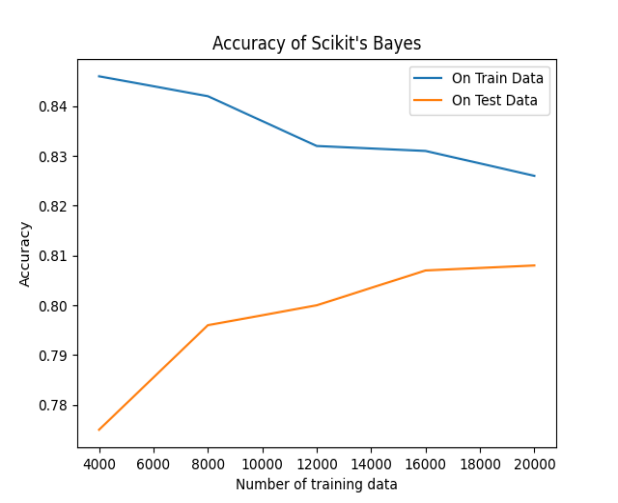
Αρχικά πίνακες και διαγράμματα για τον αλγόριθμο Naïve Bayes του SciKitLearn:

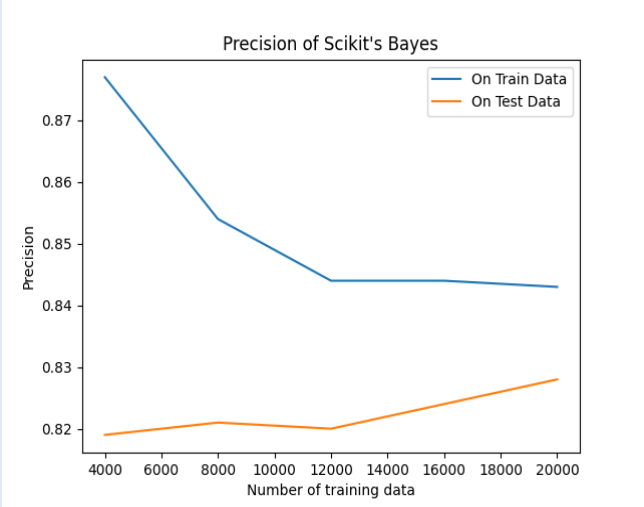
	Accuracy	
No. of Training Data	Train	Test
4000	0.846	0.775
8000	0.842	0.796
12000	0.832	0.8
16000	0.831	0.807
20000	0.826	0.808

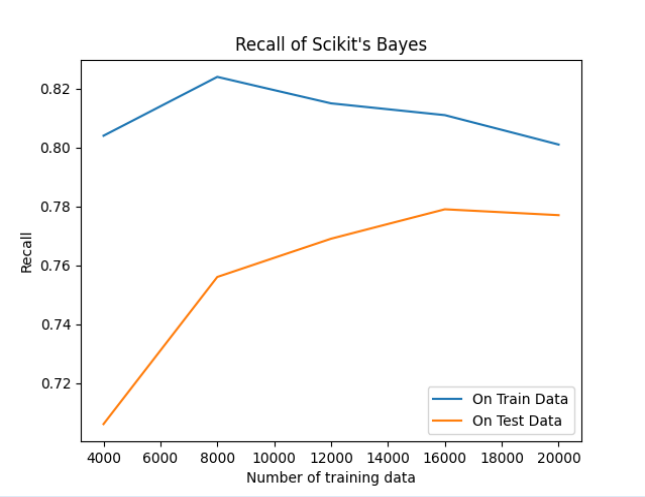
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.877	0.819
8000	0.854	0.821
12000	0.844	0.82
16000	0.844	0.824
20000	0.843	0.828

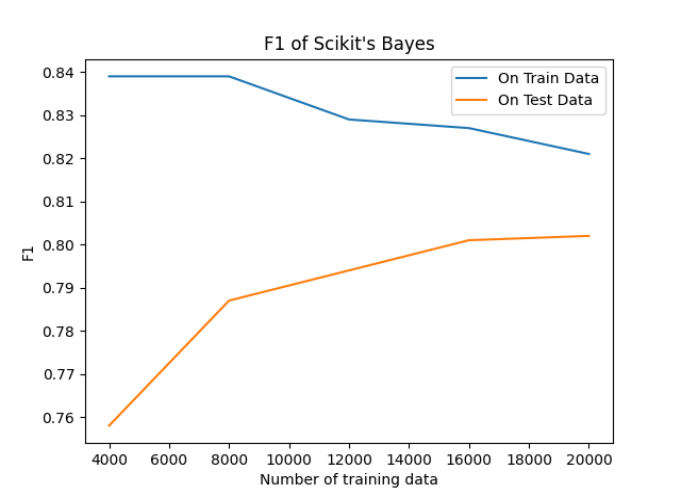
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.804	0.706
8000	0.824	0.756
12000	0.815	0.769
16000	0.811	0.779
20000	0.801	0.777

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.839	0.758
8000	0.839	0.787
12000	0.829	0.794
16000	0.827	0.801
20000	0.821	0.802

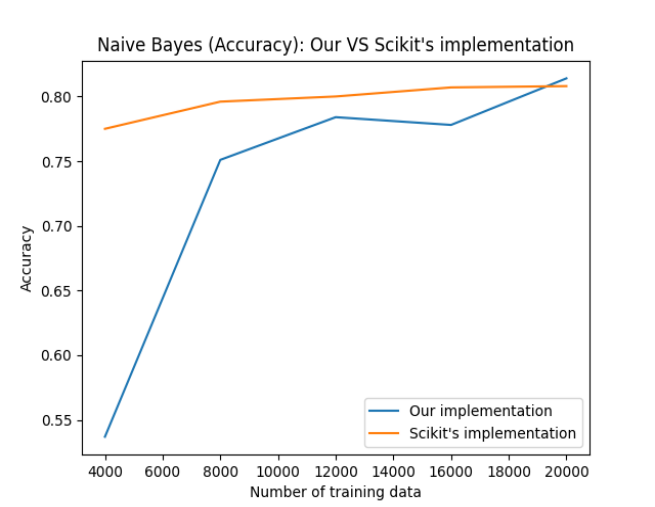


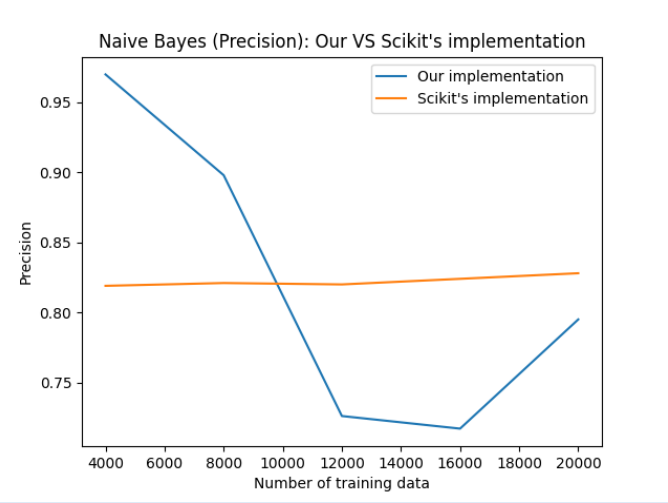


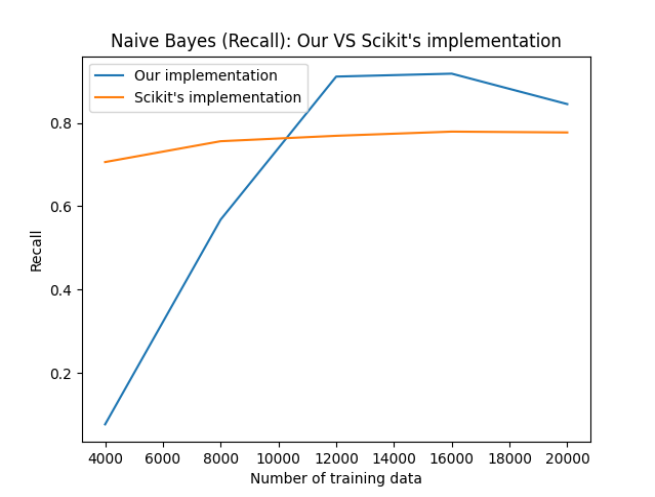


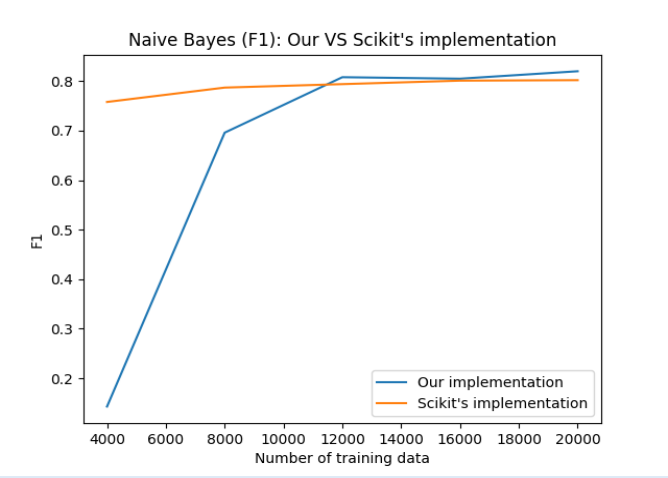


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα ελέγχου (TestData):

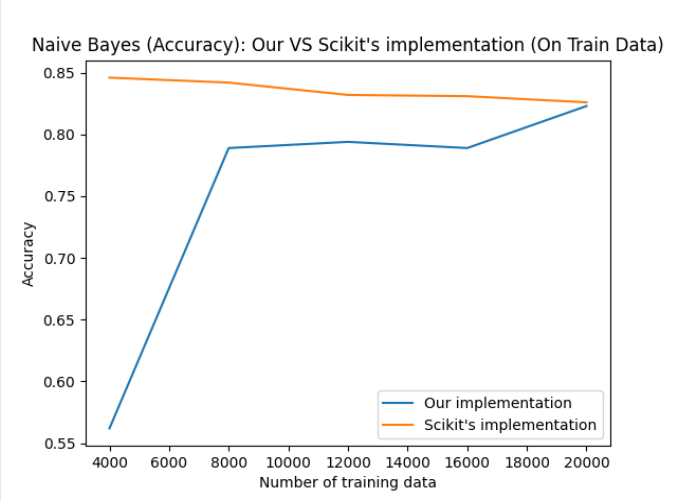


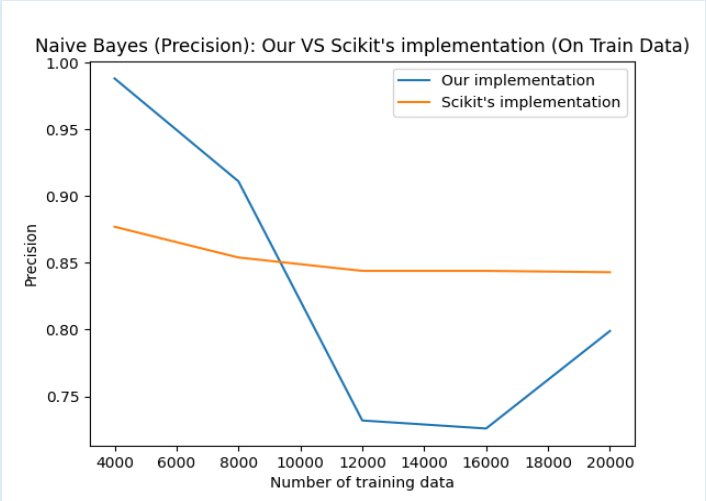


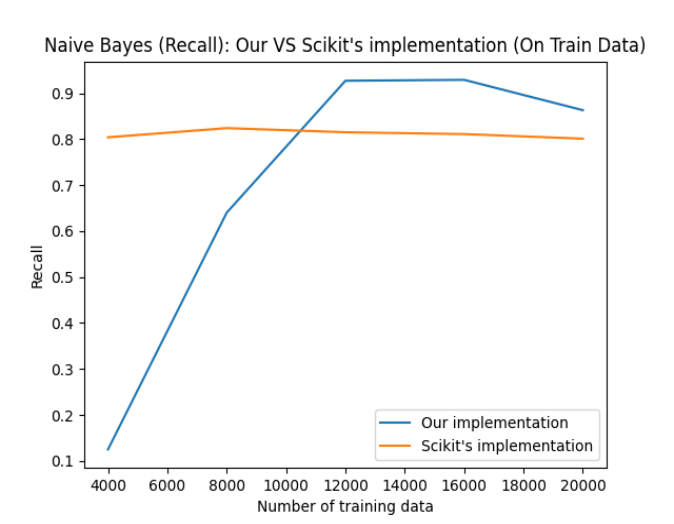


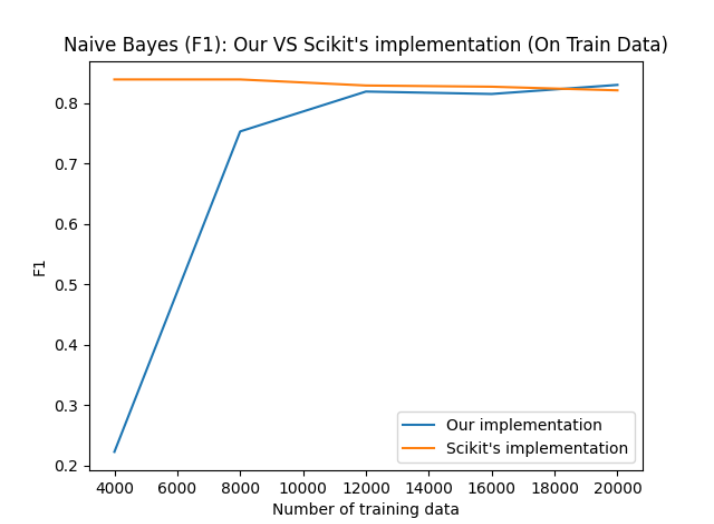


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα εκπαίδευσης (TrainData):









Σύγκριση Random Forest (Our implementation vs SciKit)

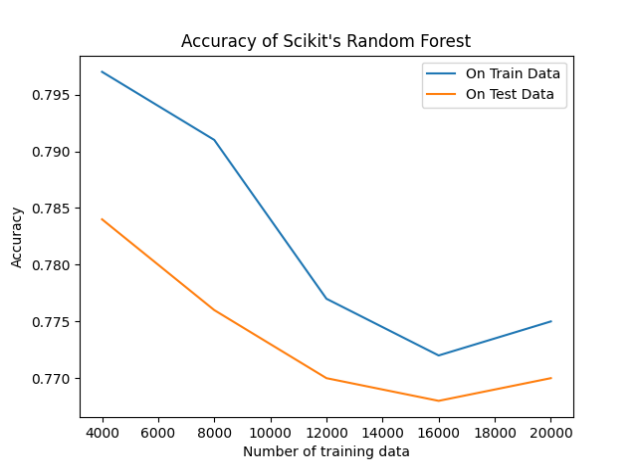
Αρχικά πίνακες και διαγράμματα για τον αλγόριθμο Random Forest του SciKitLearn:

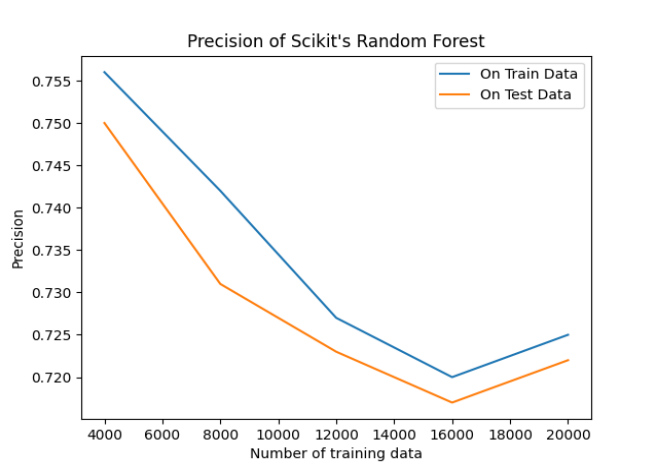
	Accuracy	
No. of Training Data	Train	Test
4000	0.797	0.784
8000	0.791	0.776
12000	0.777	0.77
16000	0.772	0.768
20000	0.775	0.77

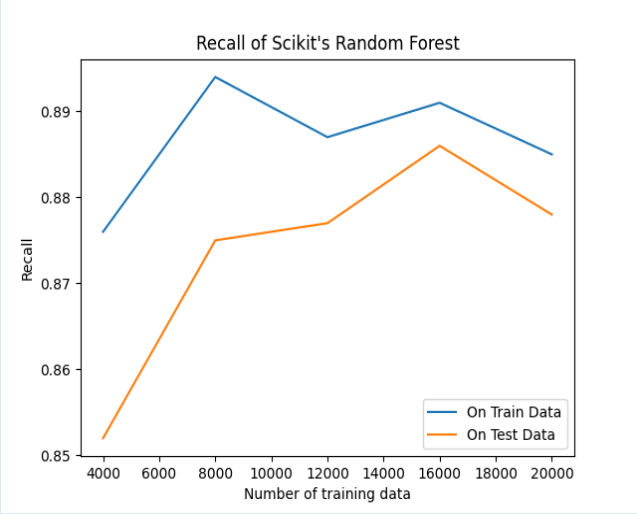
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.756	0.75
8000	0.742	0.731
12000	0.727	0.723
16000	0.72	0.717
20000	0.725	0.722

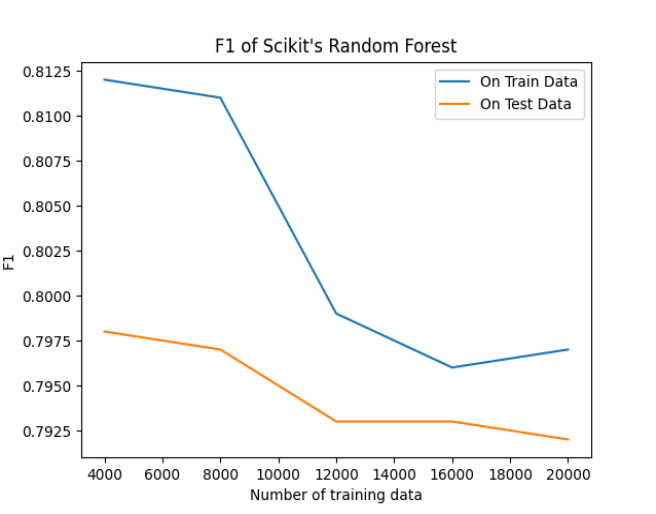
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.876	0.852
8000	0.894	0.875
12000	0.887	0.877
16000	0.891	0.886
20000	0.885	0.878

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.812	0.798
8000	0.811	0.797
12000	0.799	0.793
16000	0.796	0.793
20000	0.797	0.792

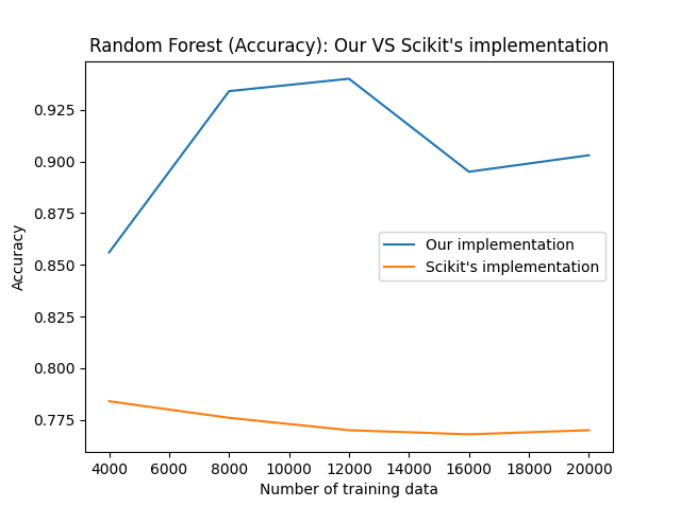


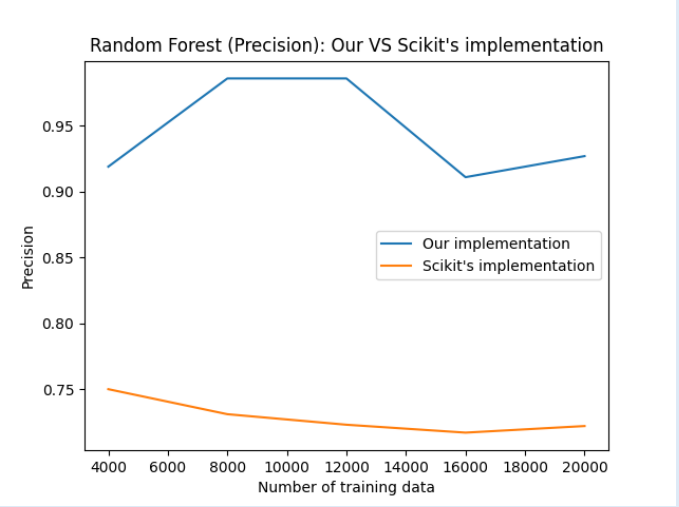


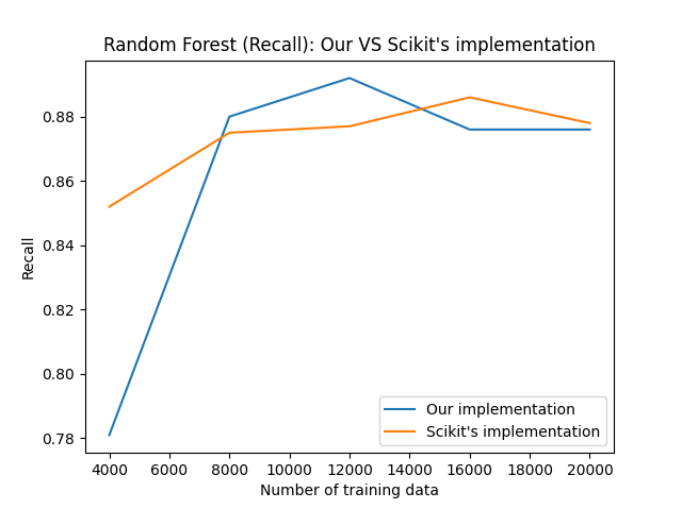


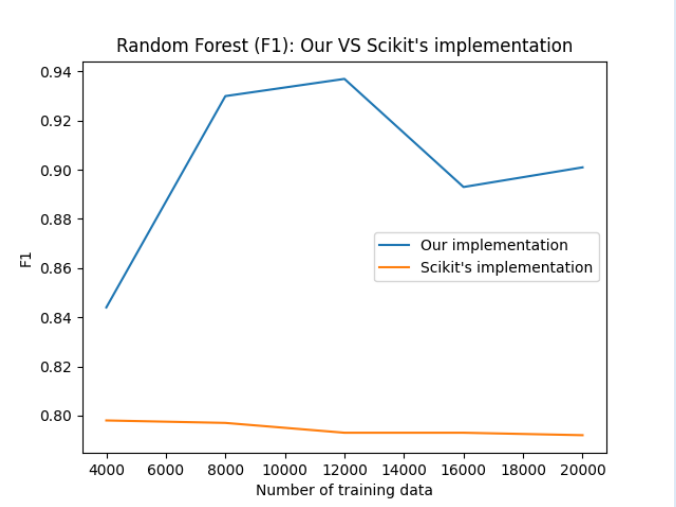


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα ελέγχου (TestData):

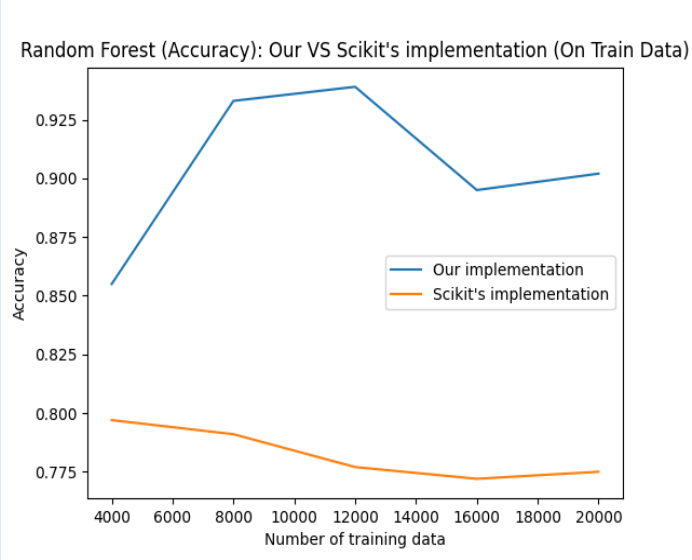


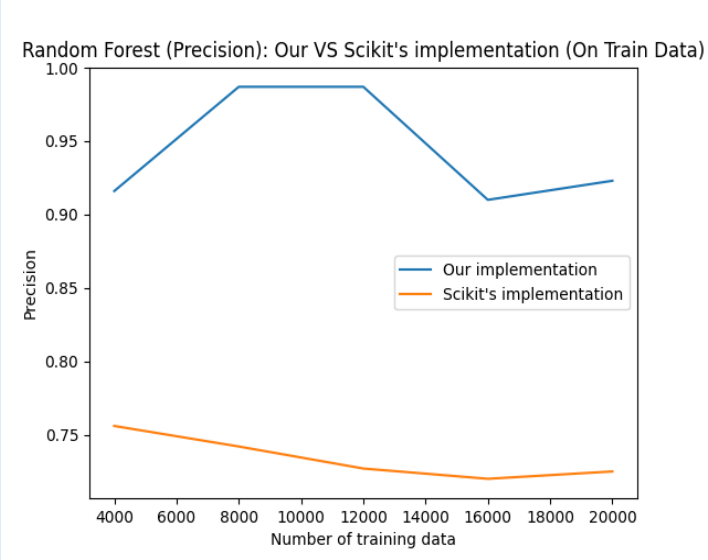


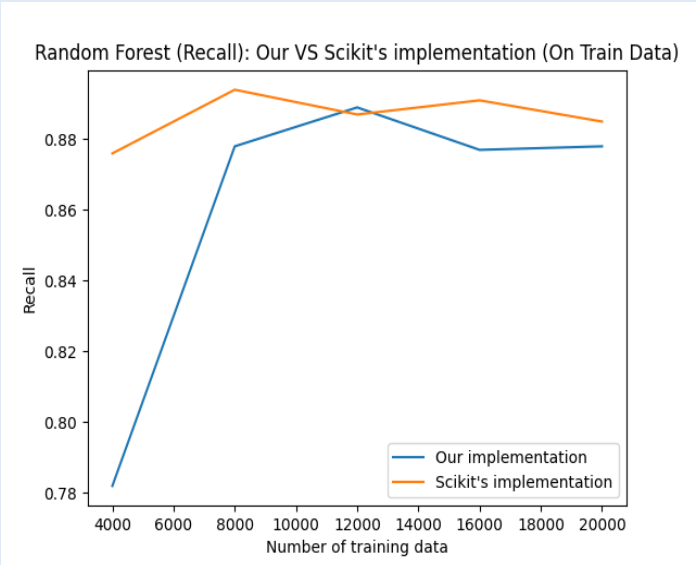


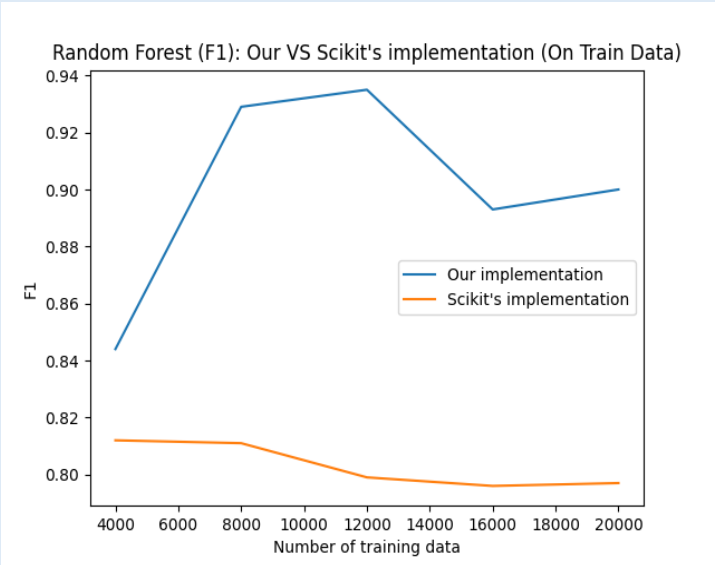


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα εκπαίδευσης (TrainData):









Σύγκριση Ada Boost (Our implementation vs SciKit)

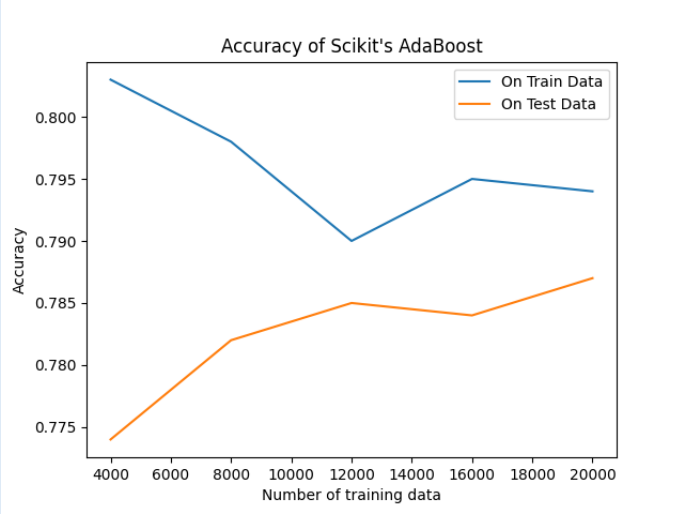
Αρχικά πίνακες και διαγράμματα για τον αλγόριθμο Ada Boost του SciKitLearn:

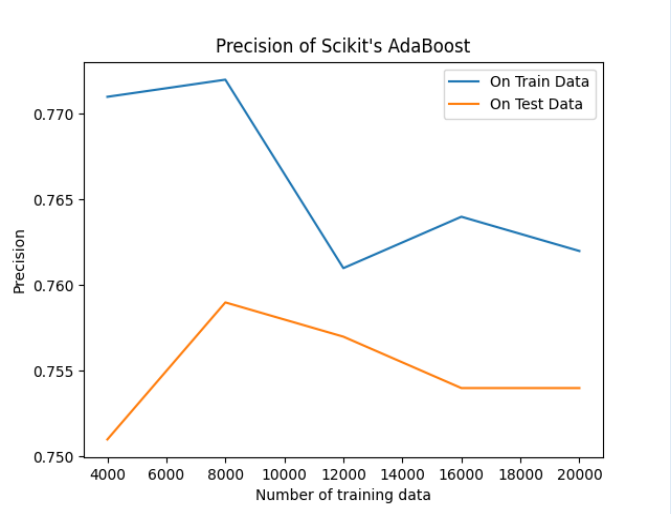
	Accuracy	
No. of Training Data	Train	Test
4000	0.803	0.774
8000	0.798	0.782
12000	0.79	0.785
16000	0.795	0.784
20000	0.794	0.787

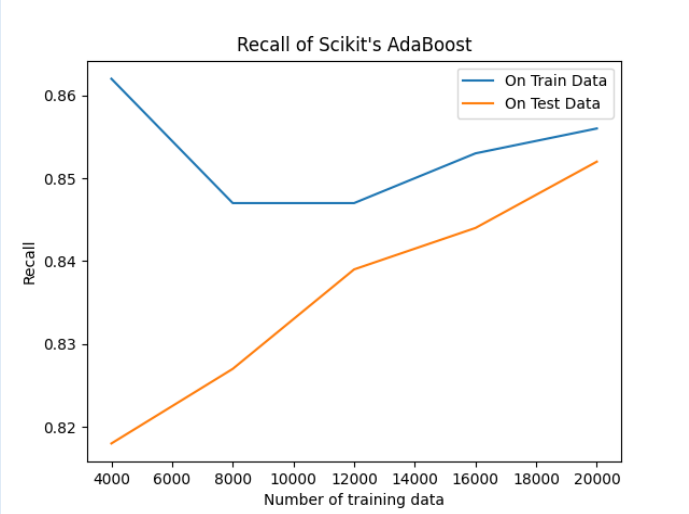
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.771	0.751
8000	0.772	0.759
12000	0.761	0.757
16000	0.764	0.754
20000	0.762	0.754

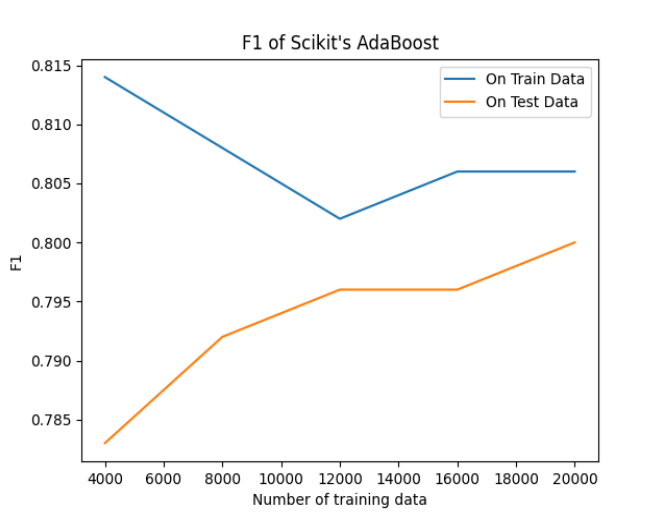
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.862	0.818
8000	0.847	0.827
12000	0.847	0.839
16000	0.853	0.844
20000	0.856	0.852

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.814	0.783
8000	0.808	0.792
12000	0.802	0.796
16000	0.806	0.796
20000	0.806	0.8

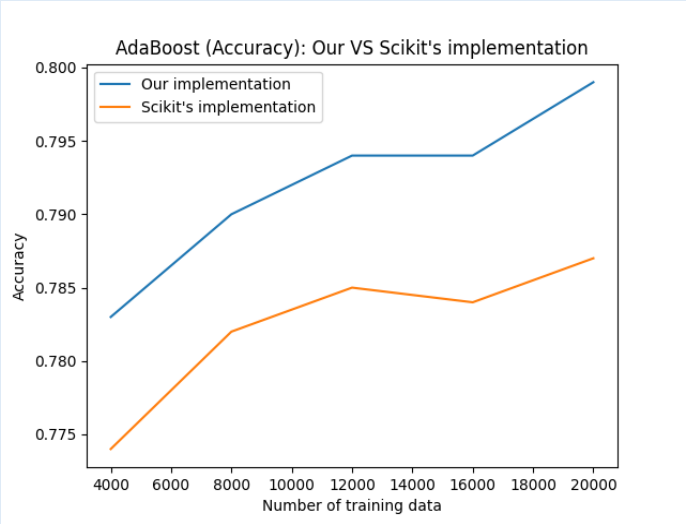


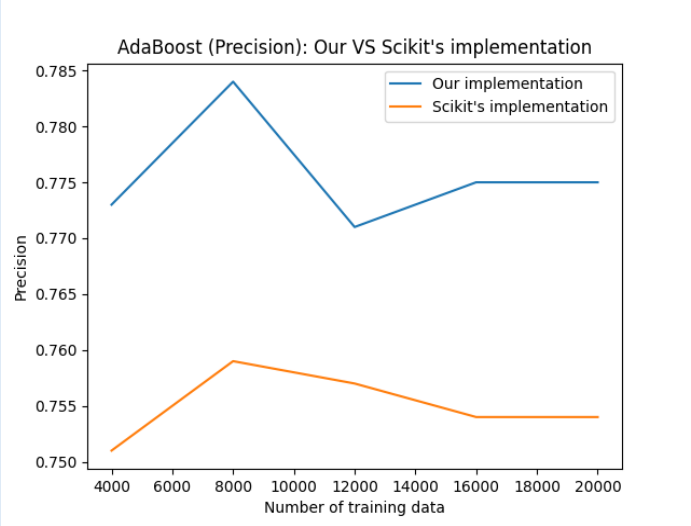


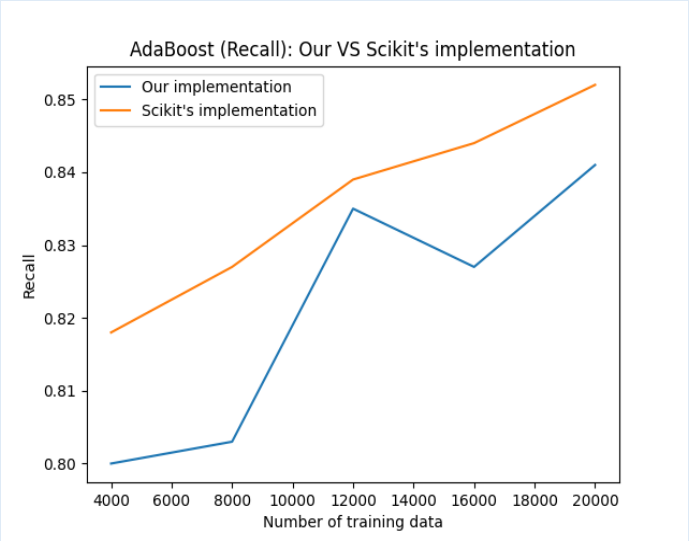


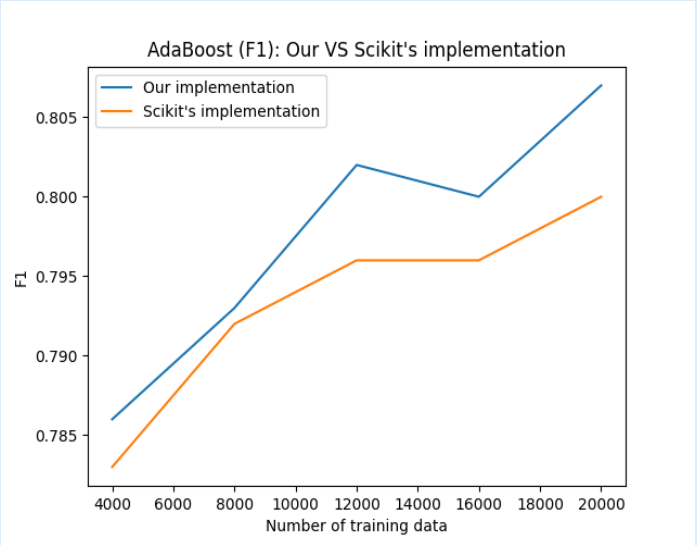


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα ελέγχου (TestData):

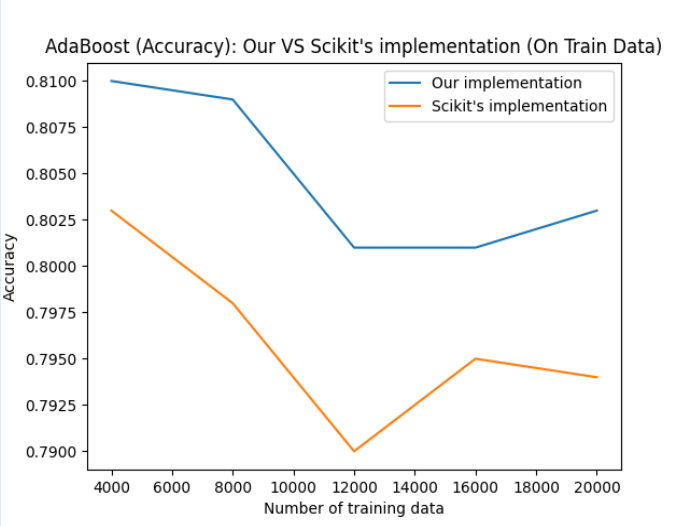


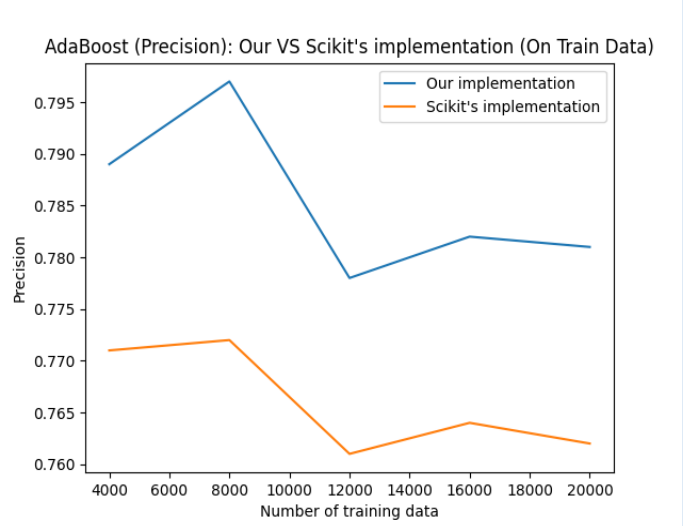


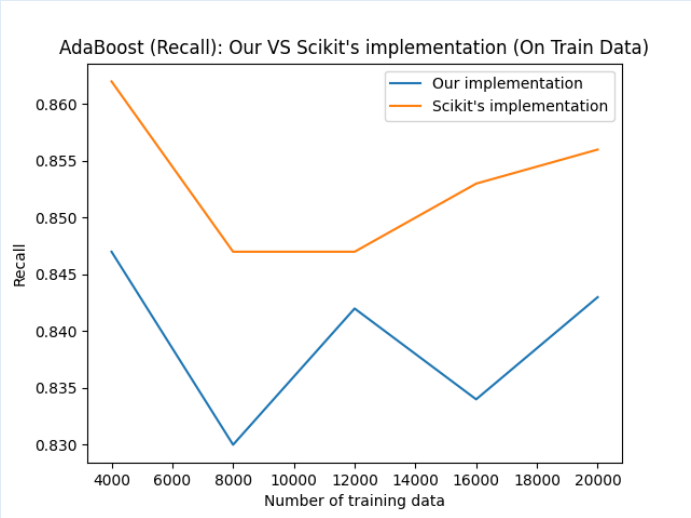


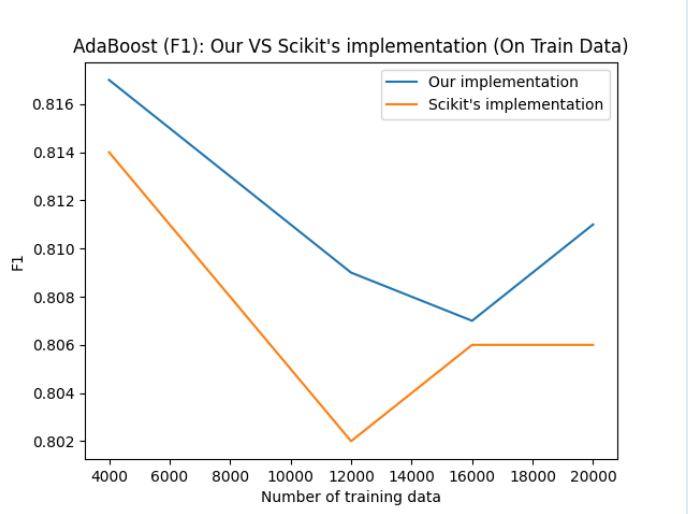


Διαγράμματα σύγκρισης της δικής μας υλοποίησης με την υλοποίηση του SciKitLearn πάνω στα δεδομένα εκπαίδευσης (TrainData):









ΜΕΡΟΣ Γ

Σε αυτό το μέρος της εργασίας αποφασίσαμε να συγκρίνουμε τα αποτελέσματα των παραπάνω μερών με αυτά ενός RNN, ο οποίος φυσικά υλοποιήθηκε σε Tensorflow/Keras.

Αρχικά παρουσιάζουμε τις μετρήσεις μας αλλά και τα διαγράμματα σχετικά με accuracy, precision, recall και F1 σε test και train data:

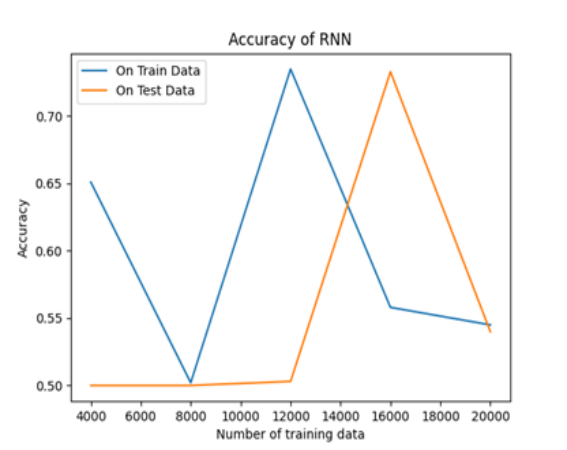
RNN

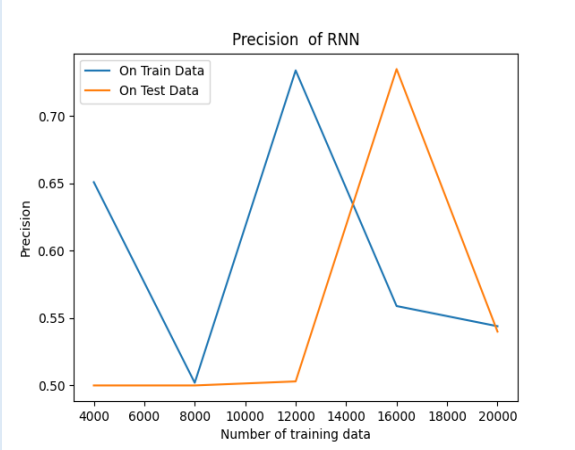
	Accuracy	
No. of Training Data	Train	Test
4000	0.651	0.5
8000	0.502	0.5
12000	0.735	0.503
16000	0.558	0.733
20000	0.545	0.54

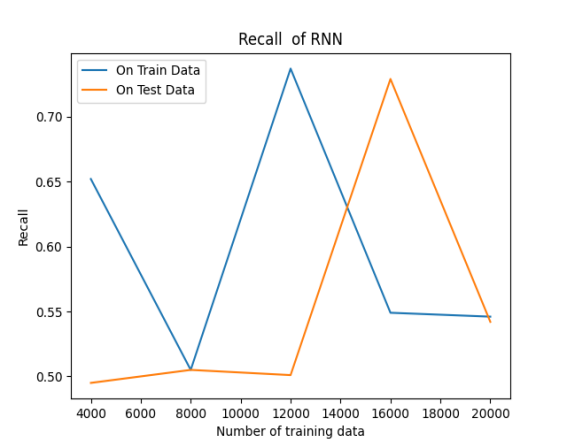
	Precision (Positive)	
No. of Training Data	Train	Test
4000	0.651	0.5
8000	0.502	0.5
12000	0.734	0.503
16000	0.559	0.735
20000	0.544	0.54

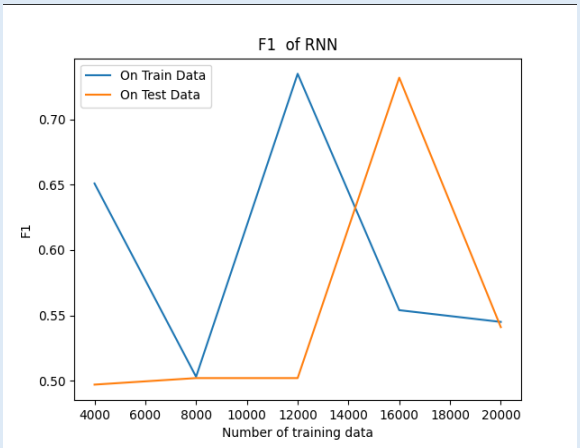
	Recall (Positive)	
No. of Training Data	Train	Test
4000	0.652	0.495
8000	0.505	0.505
12000	0.737	0.501
16000	0.549	0.729
20000	0.546	0.542

	F1 (Positive)	
No. of Training Data	Train	Test
4000	0.651	0.497
8000	0.503	0.502
12000	0.735	0.502
16000	0.554	0.732
20000	0.545	0.541









Στην συνέχεια παρουσιάζουμε τις αντίστοιχες μετρήσεις και διαγράμματα για το σφάλμα συναρτήσει του αριθμού των εποχών σε test και train data:

RNN

	Error (4000 Training Examples)	
No. Age	Train	Test
2	0.684	0.698
4	0.669	0.696
6	0.652	0.702
8	0.61	0.689
10	0.511	0.693

RNN

	Error (8000 Training Examples)	
No. Age	Train	Test
2	0.687	0.693
4	0.674	0.686
6	0.661	0.688
8	0.633	0.688
10	0.695	0.690

RNN

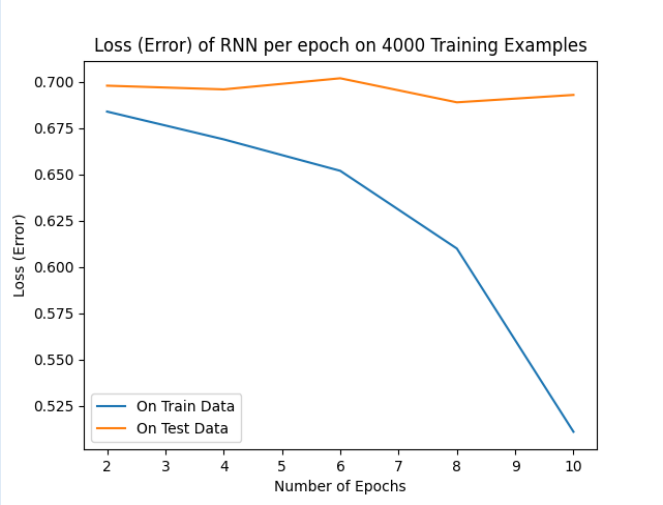
	Error (12000 Training Examples)	
No. Age	Train	Test
2	0.685	0.694
4	0.655	0.692
6	0.603	0.674
8	0.676	0.703
10	0.693	0.695

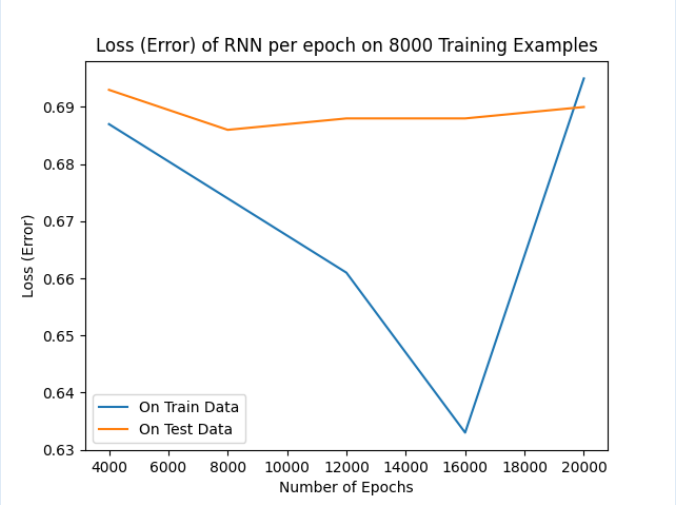
RNN

	Error (16000 Training Examples)	
No. Age	Train	Test
2	0.687	0.692
4	0.658	0.668
6	0.568	0.694
8	0.579	0.645
10	0.687	0.582

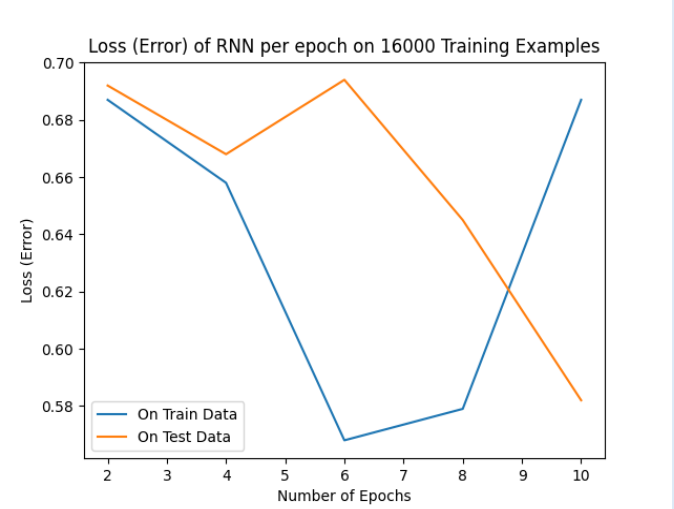
RNN

	Error (20000 Training Examples)	
No. Age	Train	Test
2	0.685	0.685
4	0.657	0.580
6	0.508	0.708
8	0.626	0.610
10	0.643	0.691


Loss (Error) of RNN per epoch on 4000 Training Examples
Loss (Error)
Number of Epochs
On Train Data
On Test Data

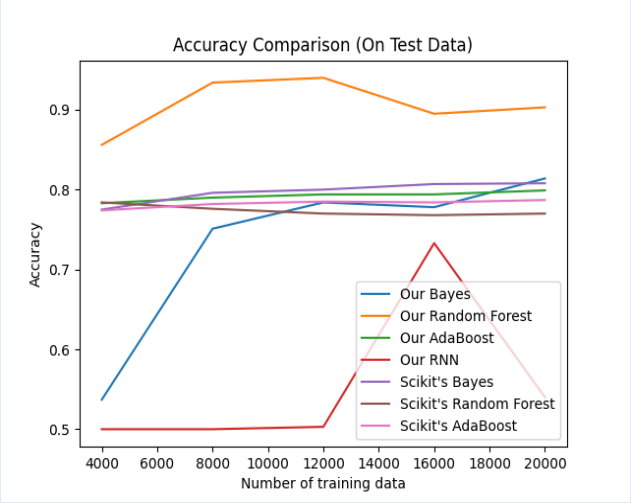

Loss (Error) of RNN per epoch on 8000 Training Examples
Loss (Error)
Number of Epochs
On Train Data
On Test Data

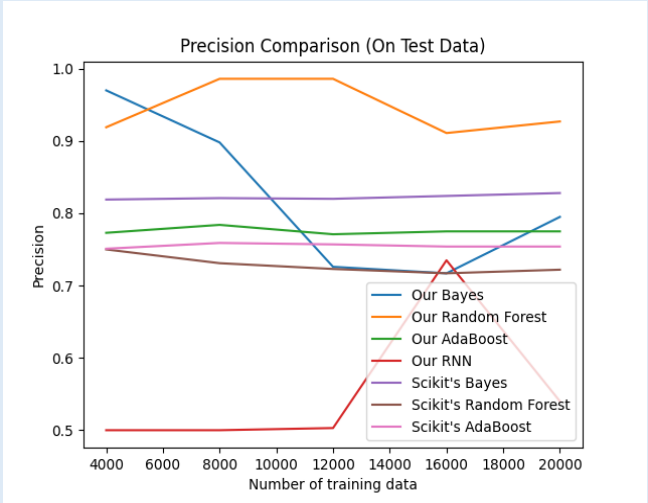

Loss (Error) of RNN per epoch on 12000 Training Examples
Loss (Error)
Number of Epochs
On Train Data
On Test Data

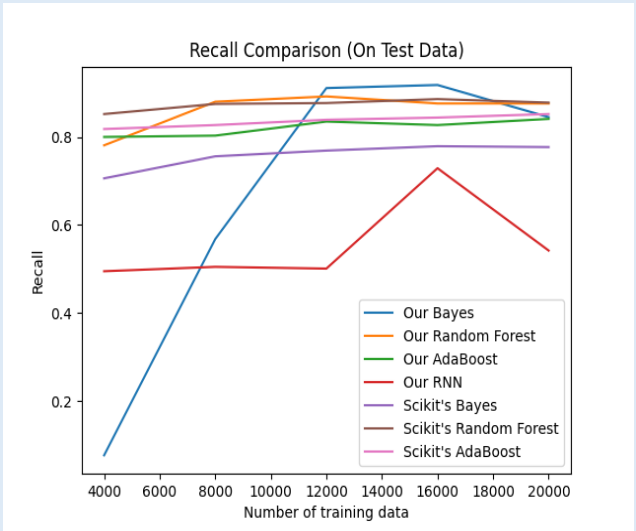

Loss (Error) of RNN per epoch on 16000 Training Examples
Loss (Error)
Number of Epochs
On Train Data
On Test Data

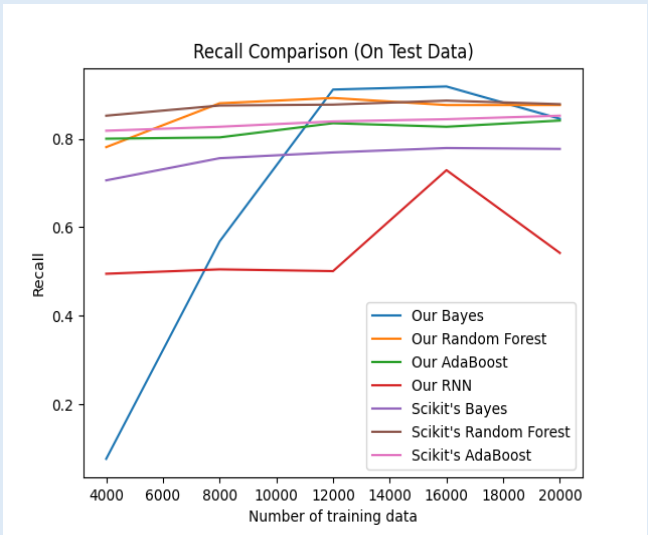

Loss (Error) of RNN per epoch on 20000 Training Examples
Loss (Error)
Number of Epochs
On Train Data
On Test Data

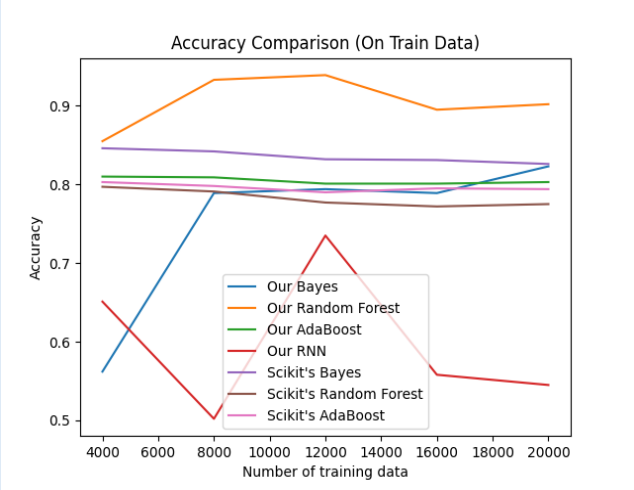
Τέλος παρουσιάζουμε τις συγκρίσεις των μετρήσεων accuracy, precision, recall και F1 των δικών μας αλγορίθμων, των αλγορίθμων του ScikitLearn με τις αντίστοιχες μετρήσεις του RNN που υλοποιήσαμε:

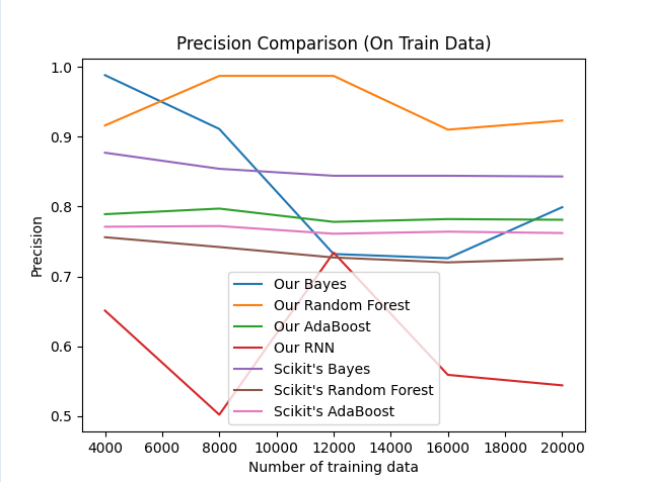




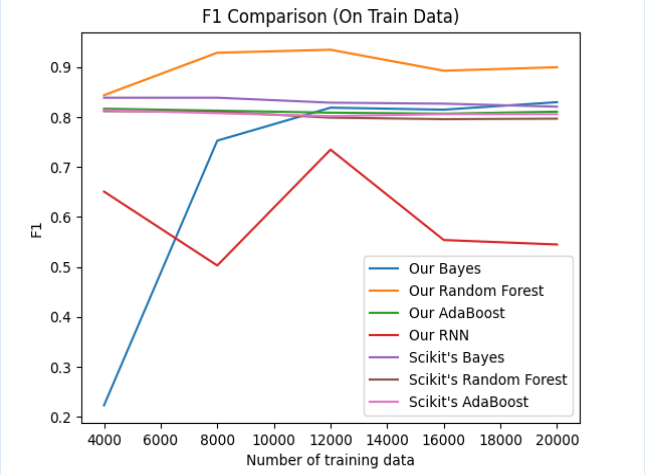





Accuracy Comparison (On Train Data)
Accuracy
Number of training data
Our Bayes
Our Random Forest
Our AdaBoost
Our RNN
Scikit's Bayes
Scikit's Random Forest
Scikit's AdaBoost


Precision Comparison (On Train Data)
Precision
Number of training data
Our Bayes
Our Random Forest
Our AdaBoost
Our RNN
Scikit's Bayes
Scikit's Random Forest
Scikit's AdaBoost


Recall Comparison (On Train Data)
Recall
Number of training data
Our Bayes
Our Random Forest
Our AdaBoost
Our RNN
Scikit's Bayes
Scikit's Random Forest
Scikit's AdaBoost


F1 Comparison (On Train Data)
F1
Number of training data
Our Bayes
Our Random Forest
Our AdaBoost
Our RNN
Scikit's Bayes
Scikit's Random Forest
Scikit's AdaBoost