ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ & ΤΕΧΝΟΛΟΓΙΑΣ ΤΗΣ ΠΛΗΡΟΦΟΡΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ 5ου ΕΞΑΜΗΝΟΥ 2024

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ

ΜΕΛΗ ΤΗΣ ΕΡΓΑΣΙΑΣ

Αθανάσιος-Ζώης Δημητρακόπουλος (Α.Μ. 3220039)

Ανδρέας Λάμπος (Α.Μ. 3220105)

ΔΙΔΑΣΚΩΝ

Ίων Ανδρουτσόπουλος

Βασικό Θέμα

Το θέμα της εργασίας είναι η υλοποίηση διαφόρων αλγορίθμων μηχανικής μάθησης της επιλογής μας (Bernoulli Naive Bayes, Logistic Regression, RNN) και χρησιμοποιώντας τους να κάνουμε sentiment analysis στις κριτικές του IMBD μέσω του IMDb Dataset καθώς επίσης και να συγκρίνουμε τα αποτελέσματα μας με αυτά των έτοιμων αλγορίθμων του πακέτου scikit-learn.

Επεξεργασία δεδομένων

Για να φορτώσουμε τα δεδομένα μας χρησιμοποιούμε τον κώδικα του φροντιστηρίου του μαθήματος όπου χρησιμοποιώντας την εντολή load_data του keras και με τις κατάλληλες τιμές m, n, k[(1)] στα ορίσματα num_words και skip_top χωρίσαμε τις μισές κριτικές με τα labels τους στα train δεδομένα και τις υπόλοιπες στα test. Πιο συγκεκριμένα, για τους αλγορίθμους που χρειάζονται validation set για το fine tuning των υπερπαραμέτρων όπως ο Logistic Regression, χωρίσαμε εκ νέου το training set στα δύο δίνοντας ένα μικρό ποσοστό γύρω στο 20% στα dev δεδομένα. Ειδικότερα, μετά από δοκιμές στον κάθε αλγόριθμο καταλήξαμε στις ακόλουθες τιμές των υπερπαραμέτρων :

1. Bernoulli Naive Bayes : m = 2500, n = 200, k = 0
2. Logistic Regression : m = 3000, n = 30, k = 20
3. RNN : m = 1000, k = 20

Σε αυτό το σημείο τα δεδομένα έχουν την μορφή κειμένου οπότε για να μπορέσουμε να τα χρησιμοποιήσουμε σε έναν binary classifier χρησιμοποιούμε τον CountVectorizer, όπως και στο φροντιστήριο, για να τα μετατρέψουμε σε πίνακες από άσσους και μηδενικά όπου το κάθε στοιχείο τους δηλώνει την ύπαρξη(1) ή όχι(0) του feature, δηλαδή της λέξης της στήλης την οποία κοιτάμε, στην κριτική της γραμμής που βρισκόμαστε.

(1) Η μεταβλητή m είναι ο αριθμός των λέξεων του λεξιλογίου και οι n και η k ο αριθμός των περισσότερο και λιγότερο συχνών λέξεων που θα προσπεράσουμε.

Bernoulli Naive Bayes

1. Υλοποίηση

Για την υλοποίηση του Bernoulli Naive Bayes έχουμε την κλάση

BernoulliNaiveBayes() η οποία έχει τις ακόλουθες μεθόδους :

a) **__init__(self):** Δεν έχει κάποια χρήση, λόγω του γεγονότος ότι η υλοποίηση μας δεν δέχεται ως ορίσματα υπερπαραμέτρους, και υπάρχει αποκλειστικά για λόγους πληρότητας.

b) **fit(self, X, y):** Δέχεται σαν όρισμα ένα feature matrix$^{(2)}$ X και ένα target vector y$^{(2)}$ και ουσιαστικά εκπαιδεύει το μοντέλο μας. Ειδικότερα, υπολογίζει αρχικά τις 2 "prior" πιθανότητες, δηλαδή την πιθανότητα μία κριτική να ανήκει στην θετική κατηγορία και την πιθανότητα να ανήκει στην αρνητική. Την πρώτη (θετική) την βρίσκουμε αθροίζοντας όλους τους άσσους του target vector και διαιρώντας με το συνολικό μήκος του και η δεύτερη είναι απλά η διαφορά της πρώτης από το 1. Στη συνέχεια, υπολογίζουμε την πιθανότητα για κάθε feature να έχει την τιμή 1 και την πιθανότητα να έχει τιμή 0 με δεδομένο ότι ανήκει σε μία κατηγορία. Επομένως, έχουμε να υπολογίσουμε την πιθανότητα για (xi = 1 και c = 1), (xi = 0 και c = 1) , (xi = 0 και c = 0) και (xi = 1 και c = 0). Η πρώτη και η τρίτη που αφορούν την ύπαρξη του feature Xi με δεδομένη την κάθε κατηγορία υπολογίζονται κρατώντας από το feature matrix X μόνο τις γραμμές που έχουν τιμή 1 και τιμή 0 αντίστοιχα στο target vector και αθροίζοντας κατακόρυφα τις γραμμές του matrix ώστε να πάρουμε ένα νέο vector μήκους όσο και οι στηλες του X το οποίο θα δείχνει σε κάθε index i τις εμφανίσεις του feature Xi για την κάθε κατηγορία .Τέλος, αν διαιρέσουμε αυτό το vector με το πλήθος των παραδειγμάτων κάθε κατηγορίας θα λάβουμε τις ζητούμενες πιθανότητες. Αντίστοιχα για να βρούμε τις άλλες δύο, δεύτερη και τέταρτη, ουσιαστικά παίρνουμε τις ήδη υπολογισμένες και τις αφαιρούμε από την μονάδα. Σε αυτό το σημείο αξίζει να σημειωθεί ότι για τον υπολογισμό των πιθανοτήτων χρησιμοποιούμε την τεχνική του Laplace smoothing για να αποφύγουμε την ύπαρξη μηδενικών πιθανοτήτων και κρατάμε

επίσης τους λογαρίθμους των πιθανοτήτων για μεγαλύτερη σταθερότητα στις πράξεις (αποφυγή πολλαπλασιασμών με πολύ μικρούς αριθμούς). c) δέχεται σαν όρισμα μόνο ένα feature matrix X και με βάση αυτό και τις πιθανότητες που έχει ήδη υπολογίσει προσπαθεί να κατηγοριοποιήσει την κάθε γραμμή του X, δηλαδή την κάθε κριτική, σε μία από τις δύο κατηγορίες. Για να το πετύχουμε αυτό υπολογίζουμε την πιθανότητα κάθε κριτικής να ανήκει στην θετική και στην αρνητική κατηγορία με δεδομένα τα features τους. Το πρώτο γίνεται αθροίζοντας το γινόμενο του matrix X με το πρώτο διάνυσμα πιθανοτήτων που υπολογίσαμε παραπάνω (περίπτωση των θέσεων του X που έχουν τιμή 1) και το γινόμενο του matrix 1-X, ο οποίος είναι ο ίδιος με τον X με την διαφορά ότι εκεί που ο X έχει τιμή 1 ο 1-X έχει 0 και το αντίστροφο, με το δεύτερο διάνυσμα πιθανοτήτων που υπολογίσαμε παραπάνω (περίπτωση των θέσεων του X που έχουν τιμή 0). Σε αυτό το σημείο έχουμε ένα διάνυσμα όσο και οι γραμμές του X και του προσθέτουμε σε κάθε γραμμή την prior πιθανότητα της θετικής κατηγορίας ώστε τώρα η κάθε γραμμή να λέει την πιθανότητα η κριτική της γραμμής να ανήκει στην θετική κατηγορία. Κάνουμε την ίδια δουλειά και για την αρνητική κατηγορία και επιστρέφουμε ένα διάνυσμα όπου η κάθε γραμμή του έχει τιμή 1 αν η πιθανότητα της θετικής κατηγορίας είναι μεγαλύτερη από την αρνητική για την συγκεκριμένη κριτική και 0 στην αντίθετη περίπτωση. 2. Αποτελέσματα (Metrics Table, Confusion Matrix, Learning Curves) Αφού κάνουμε train τον αλγόριθμο μας σε 5 splits μέσω της βοηθητικής μεθόδου evaluate estimator έχουμε τα ακόλουθα αποτελέσματα, τα οποία φαίνονται αρκετά ικανοποιητικά με υψηλό accuracy και f1 score στα test δεδομένα, αρκετά καλή ισορροπία μεταξύ του recall και του precision και χωρίς ενδείξεις για overfitting στα train δεδομένα.

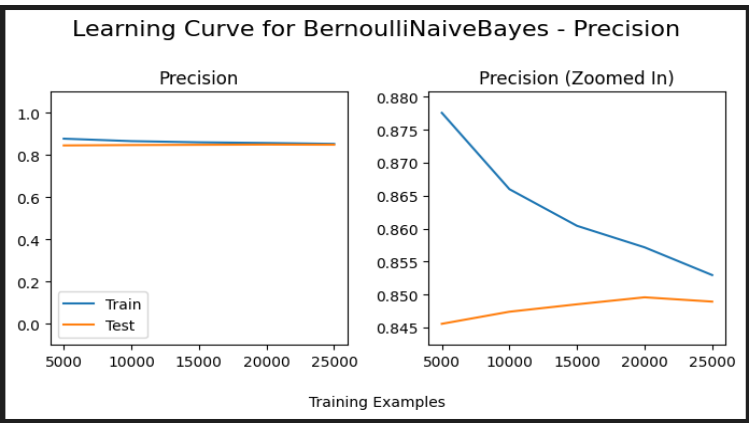
c) **predict(self, X):** δέχεται σαν όρισμα μόνο ένα feature matrix X και με βάση αυτό και τις πιθανότητες που έχει ήδη υπολογίσει προσπαθεί να κατηγοριοποιήσει την κάθε γραμμή του X, δηλαδή την κάθε κριτική, σε μία από τις δύο κατηγορίες. Για να το πετύχουμε αυτό υπολογίζουμε την πιθανότητα κάθε κριτικής να

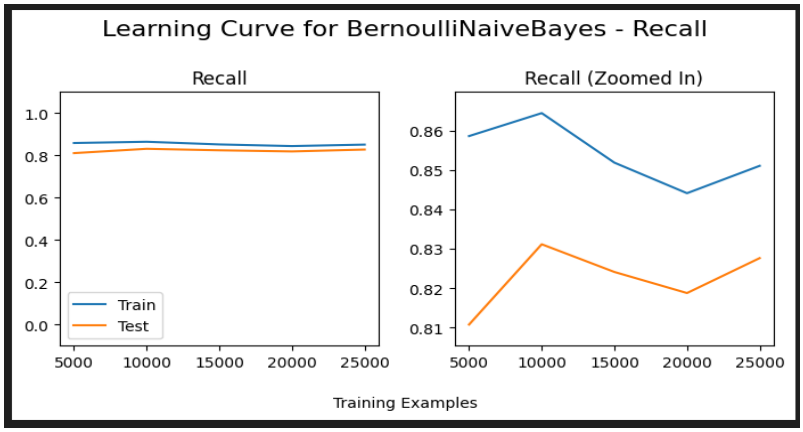
ανήκει στην θετική και στην αρνητική κατηγορία με δεδομένα τα features τους. Το πρώτο γίνεται αθροίζοντας το γινόμενο του matrix X με το πρώτο διάνυσμα πιθανοτήτων που υπολογίσαμε παραπάνω (περίπτωση των θέσεων του X που έχουν τιμή 1) και το γινόμενο του matrix 1-X, ο οποίος είναι ο ίδιος με τον X με την διαφορά ότι εκεί που ο X έχει τιμή 1 ο 1-X έχει 0 και το αντίστροφο, με το δεύτερο διάνυσμα πιθανοτήτων που υπολογίσαμε παραπάνω (περίπτωση των θέσεων του X που έχουν τιμή 0). Σε αυτό το σημείο έχουμε ένα διάνυσμα όσο και οι γραμμές του X και του προσθέτουμε σε κάθε γραμμή την prior πιθανότητα της θετικής κατηγορίας ώστε τώρα η κάθε γραμμή να λέει την πιθανότητα η κριτική της γραμμής να ανήκει στην θετική κατηγορία. Κάνουμε την ίδια δουλειά και για την αρνητική κατηγορία και επιστρέφουμε ένα διάνυσμα όπου η κάθε γραμμή του έχει τιμή 1 αν η πιθανότητα της θετικής κατηγορίας είναι μεγαλύτερη από την αρνητική για την συγκεκριμένη κριτική και 0 στην αντίθετη περίπτωση.

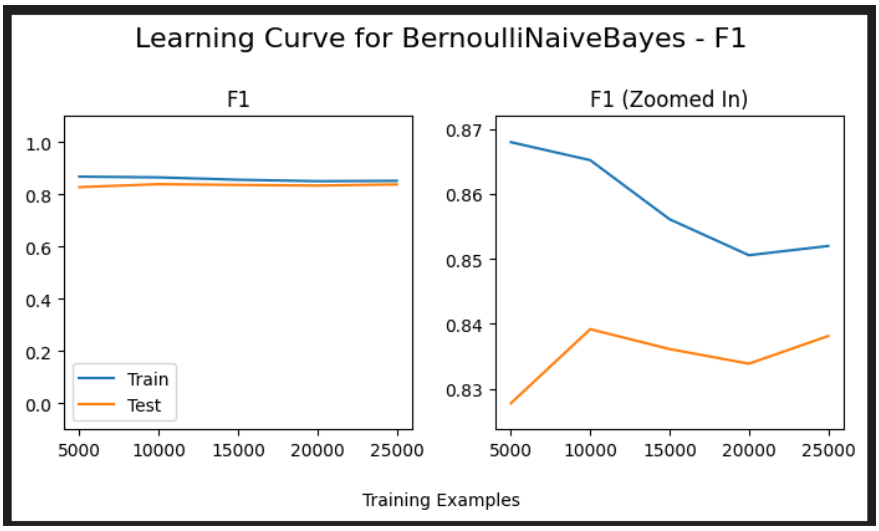
2. Αποτελέσματα (Metrics Table, Confusion Matrix, Learning Curves)

Αφού κάνουμε train τον αλγόριθμο μας σε 5 splits μέσω της βοηθητικής μεθόδου evaluate estimator έχουμε τα ακόλουθα αποτελέσματα, τα οποία φαίνονται αρκετά ικανοποιητικά με υψηλό accuracy και f1 score στα test δεδομένα, αρκετά καλή ισορροπία μεταξύ του recall και του precision και χωρίς ενδείξεις για overfitting στα train δεδομένα.

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.88	0.85	0.86	0.81	0.87	0.83
10000	0.87	0.85	0.86	0.83	0.87	0.84
15000	0.86	0.85	0.85	0.82	0.86	0.84
20000	0.86	0.85	0.84	0.82	0.85	0.83
25000	0.85	0.85	0.85	0.83	0.85	0.84





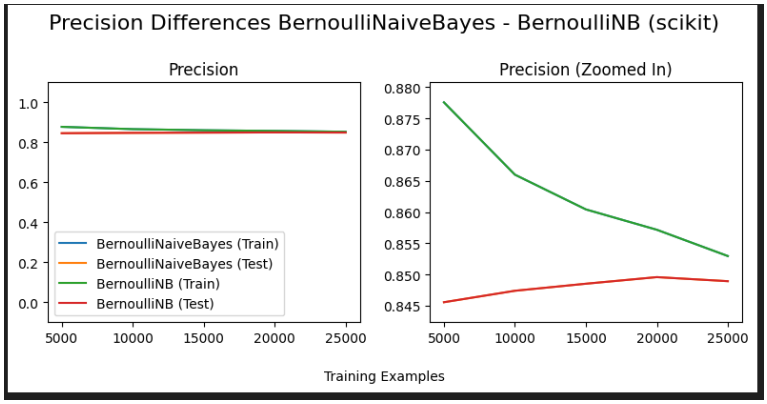


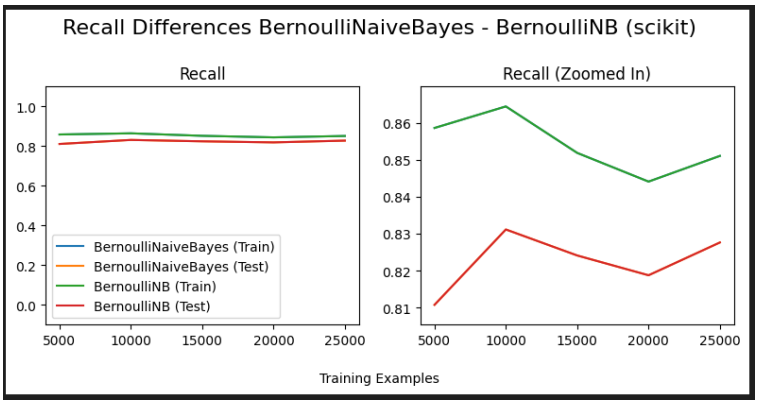
3. Συγκρίσεις με Scikit-Learn

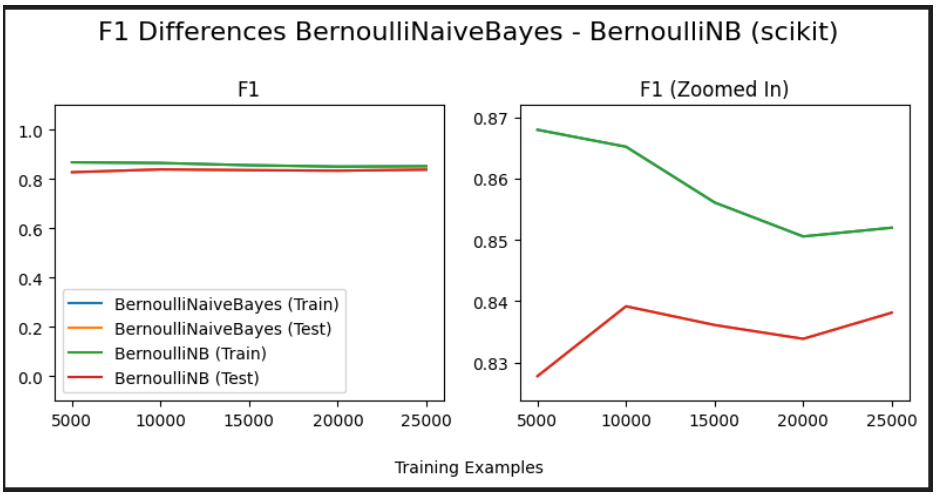
Bernoulli Naive Bayes VS Bernoulli Naive Bayes (scikit)

Μπορούμε να παρατηρήσουμε από τις παρακάτω εικόνες ότι τα αποτελεσματα της υλοποίησης μας ταυτίζονται πλήρως με αυτά της έτοιμης υλοποίησης του scikit-learn αφού ο πίνακας διαφοράς των μετρικών έχει την τιμή 0 σε όλα τα πεδία και οι καμπύλες μάθησης για όλες τις μετρικές βρίσκονται η μία πάνω στην άλλη.

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.0	0.0	0.0	0.0	0.0	0.0
10000	0.0	0.0	0.0	0.0	0.0	0.0
15000	0.0	0.0	0.0	0.0	0.0	0.0
20000	0.0	0.0	0.0	0.0	0.0	0.0
25000	0.0	0.0	0.0	0.0	0.0	0.0



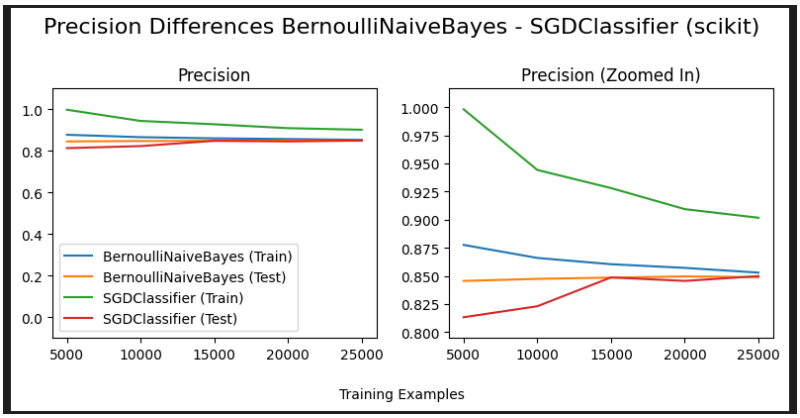


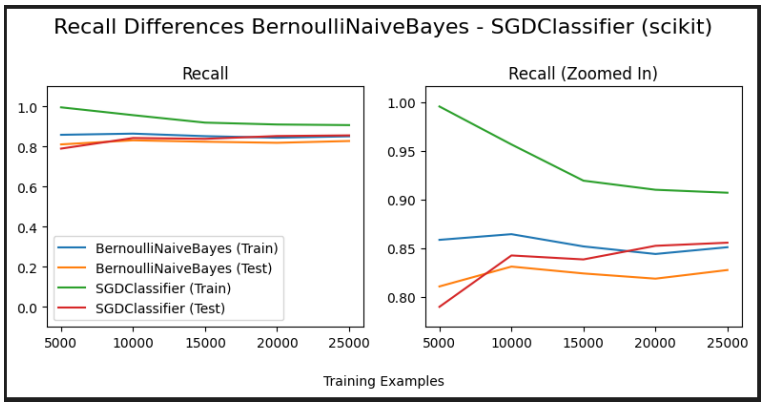


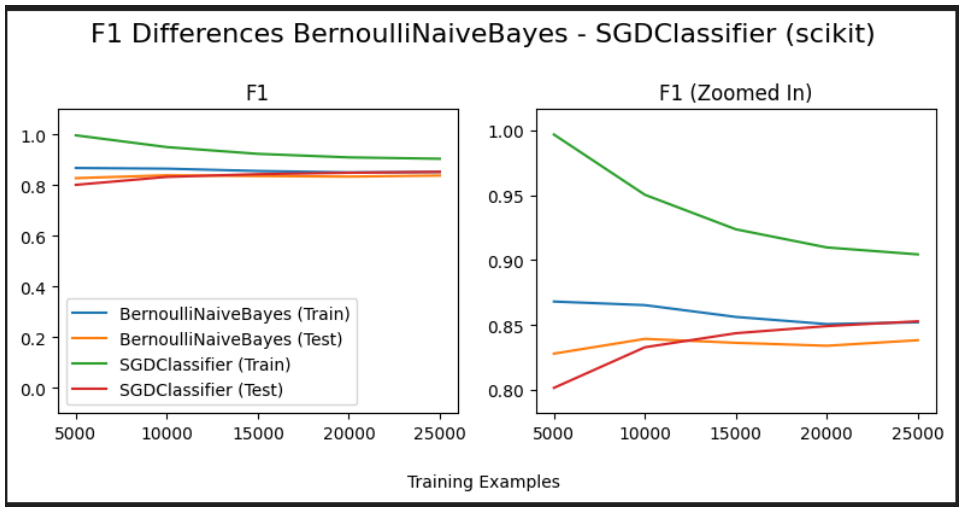
Bernoulli Naive Bayes VS Logistic Regression (scikit)

Ο αλγόριθμος μας τα πηγαίνει εξίσου καλά και ενάντια στον έτοιμο αλγόριθμο Logistic Regression με στοχαστική ανάβαση κλίσης του scikit-learn, καθώς όπως φαίνεται από τον παρακάτω πίνακα διαφοράς μετρικών πετυχαίνει σχεδόν εξίσου καλό(+/- 0.1) accuracy, precision και f1 score με πολύ μικρότερες τιμές μετρικών στα δεδομένα εκπαίδευσης. Σημαντική διαφορά παρατηρείται στην τιμή της μετρικής recall στα test δεδομένα. (Οι καμπύλες μάθησης παραλείπονται για εξοικονόμηση χώρο

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	-0.12	0.04	-0.14	0.02	-0.13	0.03
10000	-0.07	0.03	-0.10	-0.01	-0.08	0.01
15000	-0.07	0.00	-0.07	-0.02	-0.06	0.00
20000	-0.05	0.00	-0.07	-0.03	-0.06	-0.02
25000	-0.05	0.00	-0.06	-0.03	-0.05	-0.01







Logistic Regression

1. Υλοποίηση

Για τον Logistic Regression με Stohastic Gradient Descent έχουμε την κλάση **LogisticRegressionGD** με τις εξής μεθόδους:

a) **__init__(self, learning_rate=0.01, num_epochs=1000, lambda_value=0.001, threshold=0.5, tol=1e-4, patience=5):** Παίζει τον ρόλο του constructor της κλάσης αρχικοποιώντας τις παραμέτρους του μοντέλου μας.

b) **sigmoid(self,x):** Επιστρέφει το αποτέλεσμα της σιγμοειδούς συνάρτησης πάνω στο όρισμα x.

c) **initialize_parameters(self, n_features):** Αρχικοποιούμε τον πίνακα των βαρών με μηδενικά μαζί και το bias term.

d) **forward(self, X):** Υπολογίζει και επιστρέφει το αποτέλεσμα της σιγμοειδούς συνάρτησης του γραμμικού συνδυασμού του ορίσματος x με τον πίνακα των βαρών.

e) **stohasatic_gradient_descent(self, X, y):** Υλοποιεί την στοχαστική κατάβαση κλίσης κατά την οποία για κάθε γραμμή του

πίνακα X υπολογίζει την πιθανότητα να ανήκει στην θετική κατηγορίας με την forward και ενημερώνει τα βάρη σύμφωνα με τον τύπο όπου η παράγωγος της $\vec{w}_{t+1} \leftarrow \vec{w}_t - \alpha \nabla L(\vec{w}_t)$ συνάρτησης του loss (dw) έχει την μορφή : καθώς και το

$$dw = \frac{1}{m} \sum_{i=1}^{m} (y_{pred}^{(i)} - y^{(i)}) x^{(i)T} + 2\lambda w$$

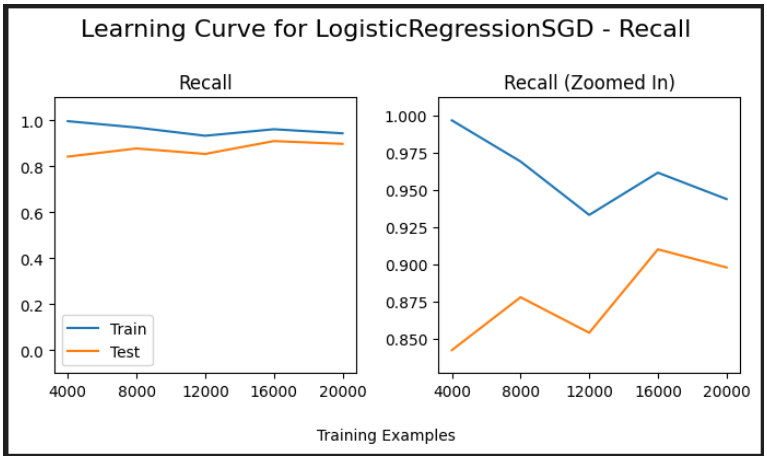
regularization term το οποίο χρησιμοποιείται για να τιμωρεί τα μεγάλα βάρη και βοηθάει στην αποφυγή του overfitting**.**

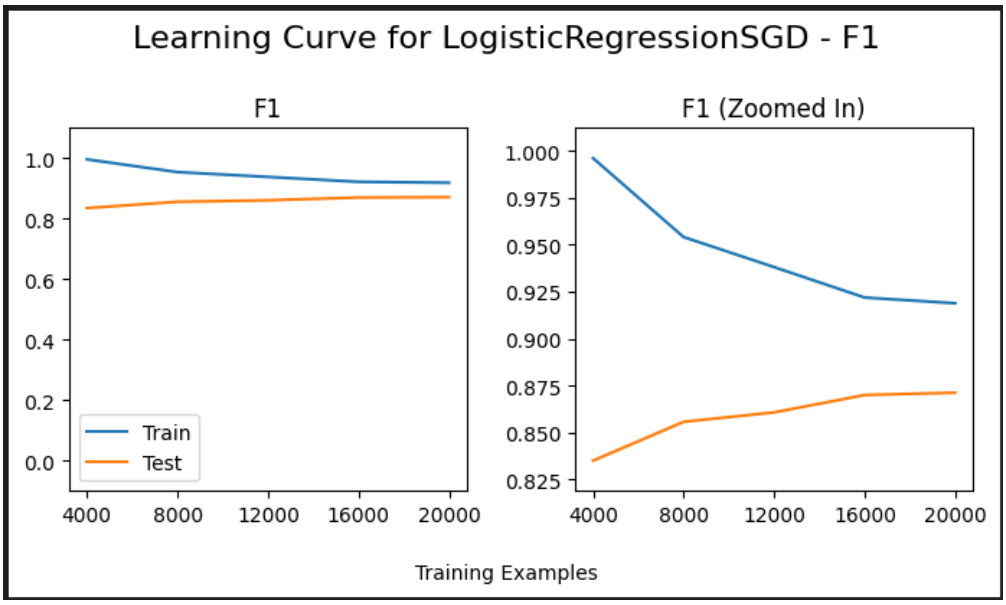
f) **compute_loss(self, y_true, y_pred):** Υπολογίζει και επιστρέφει το συνολικό error το οποίο περιλαμβάνει το cross entropy loss: καθώς και το $-\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$ regularization term το οποίο χρησιμοποιείται για να τιμωρεί τα μεγάλα βάρη και βοηθάει στην αποφυγή του overfitting.
g) **fit(self, X, y):** Είναι η μέθοδος που εκπαιδεύει το μοντέλο μας. Τρέχει εποχές μέχρι να φτάσει τον μέγιστο αριθμών εποχών (num_epochs) ή μέχρι να διαπιστωθεί ότι το μοντέλο δεν έχει βελτίωση όσο εκπαιδεύεται. Πριν τον συγκεκριμένο έλεγχο και επειδή χρησιμοποιούμε στοχαστική κατάβαση κλίσης ανακατεύουμε τα παραδείγματα με την shuffle ώστε να υπάρχει τυχαίοτητα και καλούμε την μέθοδο stohastic_gradient_descent που περιγράψαμε παραπάνω. Αφού αυτή ολοκληρωθεί υπολογίζουμε το loss της συγκεκριμένης εποχής και με βάση αυτό κάνουμε τον έλεγχο για την σύγκλιση. Αν patience (= 5) φορές βρούμε loss μεγαλύτερο από το ελάχιστο loss κατά μία μικρή τιμή tol (= 1e-4) σταματάμε. Οι τιμές patience και tol είναι υπερπαράμετροι και οι τιμές έχουν προκύψει έπειτα από δοκιμές.
h) **predict(self, X):** Για κάθε παράδειγμα εκπαίδευσης υπολογίζει την πιθανότητα να ανήκει στην θετική κατηγορία και αν αυτή είναι μεγαλύτερη από την τιμή του threshold, η οποία επίσης είναι υπερπαράμετρος, του δίνει ετικέτα, δηλαδή προβλέπει ‘1’ αλλιώς ‘0’.

2. Αποτελέσματα (Metrics Table, Confusion Matric, Learning Curves)

Όμοια με πριν εκπαιδεύσαμε τον αλγόριθμο σε 5 splits και χρησιμοποιήσαμε επίσης το 20% των training δεδομένων για το validation set. Το μοντέλο φαίνεται να τα πηγαινεί αρκετά καλά συμφωνα με τις υψηλές τιμές στα δεδομένα ελέγχου ενώ δεν υπάρχουν ενδείξεις για overfitting. Επιπλέον, για τον ταξινομητή λογιστικής παλινδρόμησης έχουμε κάνει fine-tuning σε δύο από τις υπερπαραμέτους του (lambda, threshold) ξεχωριστά επειδή η τιμή της μίας δεν επηρεάζει την άλλη. Και για τις δύο έχουμε επιλέξει ορισμένες συνηθισμένες τιμές και βλέπουμε ποια από αυτές για το 'λ' μας δίνει καλύτερο accuracy και για το threshold καλύτερο f1 score. Καταλήξαμε στις λ = 0.001 και threshold = 0.465. Αξίζει να σημειωθεί ότι λόγω της στοχαστικότητας του μοντέλου οι προβλέψεις αλλάζουν σε κάθε εκτέλεση και εμείς κρατήσαμε μια από τις πιο ικανοποιητικές.

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
4000	1.00	0.83	1.00	0.84	1.00	0.84
8000	0.94	0.83	0.97	0.88	0.95	0.86
12000	0.94	0.87	0.93	0.85	0.94	0.86
16000	0.89	0.83	0.96	0.91	0.92	0.87
20000	0.90	0.85	0.94	0.90	0.92	0.87



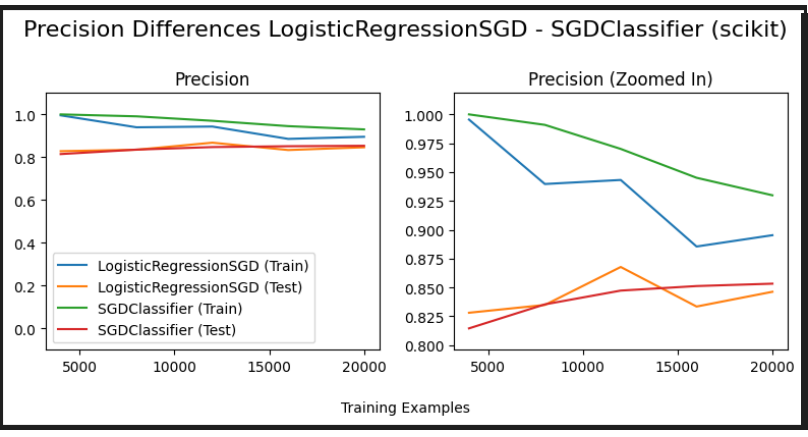


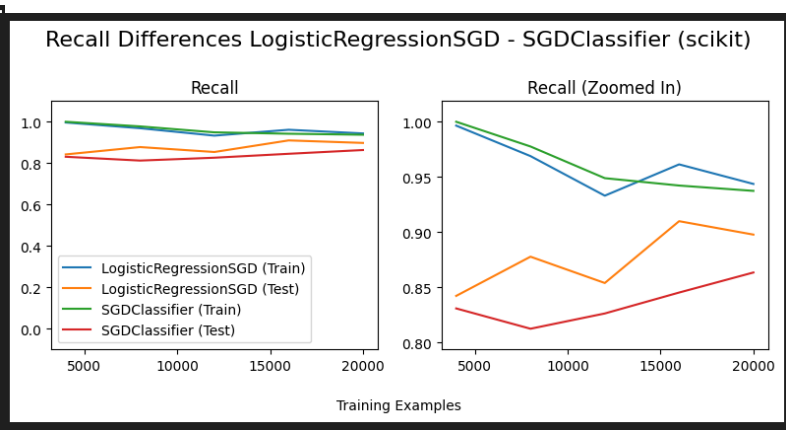
3. Συγκρίσεις με Scikit-Learn

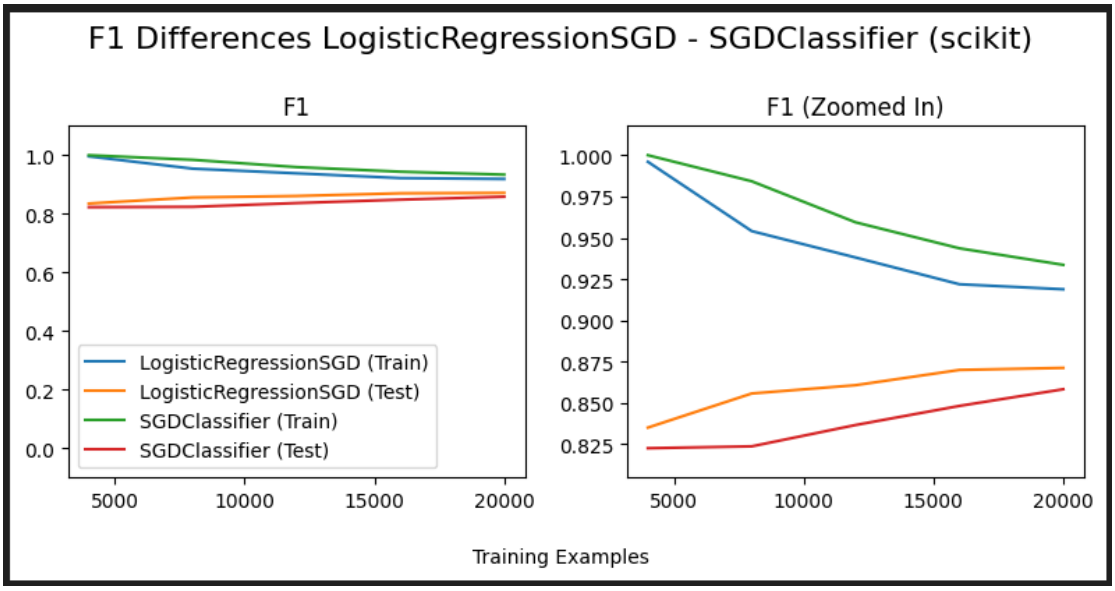
Logistic Regression VS Logistic Regression (scikit)

Το μοντέλο μας τα πηγαίνει πολύ καλά απέναντι στην έτοιμη υλοποίηση του ίδιου αλγορίθμου του scikit-learn, καθώς όπως φαίνεται από τον πίνακα διαφορών και από τις καμπύλες μάθησης πετυχαίνει μικρότερες τιμές μετρικών στα training δεδομένα και μεγαλύτερες στα testing.

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
4000	0.00	0.02	0.00	0.01	0.00	0.02
8000	-0.05	-0.01	-0.01	0.07	-0.03	0.04
12000	-0.03	0.02	-0.02	0.02	-0.02	0.02
16000	-0.06	-0.02	0.02	0.06	-0.02	0.02
20000	-0.03	0.00	0.00	0.04	-0.01	0.01



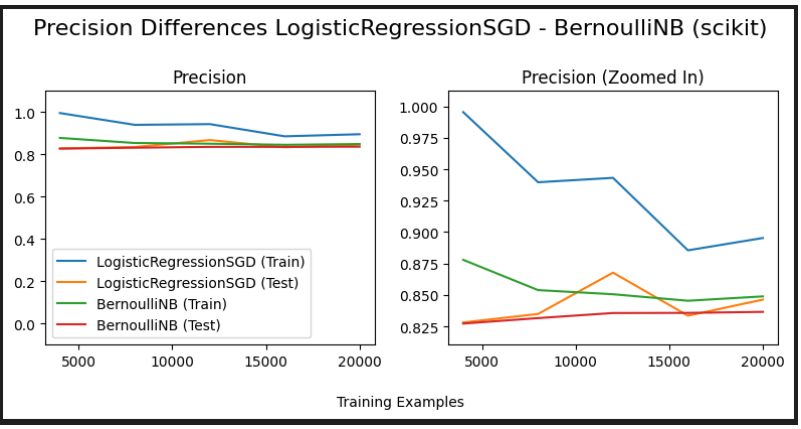


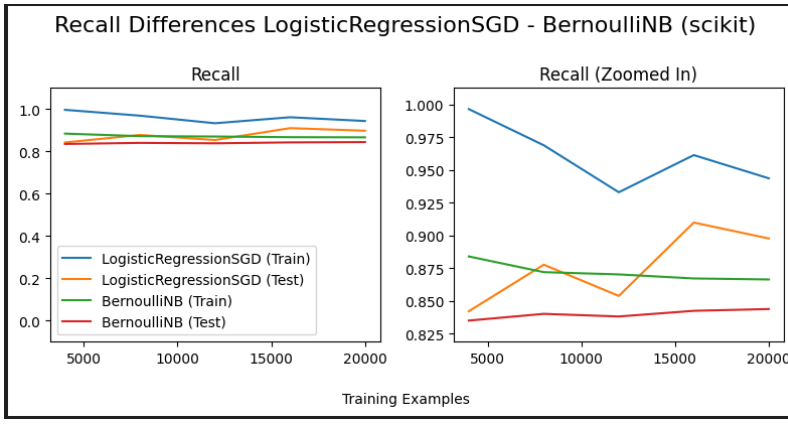


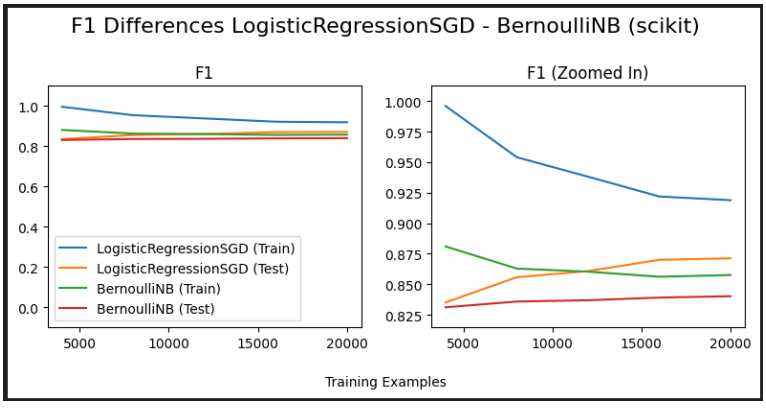
Logistic Regression VS Bernoulli Naive Bayes (scikit)

Όπως φαίνεται το μοντέλο μας ξεπερνά τον Bernoulli Naive Bayes σε όλες τις μετρικές, από τις οποίες οι υψηλότερες αλλά φυσιολογικές χωρίς overfitting τιμές στα training δεδομένα μας δείχνουν ότι το μοντέλο μάλλον είναι πιο πιθανό να γενικευτεί καλύτερα, ενώ οι υψηλότερες μετρικές στα testing δεδομένα μας απόδεικνύουν ότι όντως τα πηγαίνει καλύτερα σε νέα δεδομένα.

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
4000	0.12	0.00	0.12	0.00	0.12	0.01
8000	0.09	0.00	0.10	0.04	0.09	0.02
12000	0.09	0.03	0.06	0.01	0.08	0.02
16000	0.04	-0.01	0.09	0.07	0.06	0.03
20000	0.05	0.01	0.07	0.06	0.06	0.03







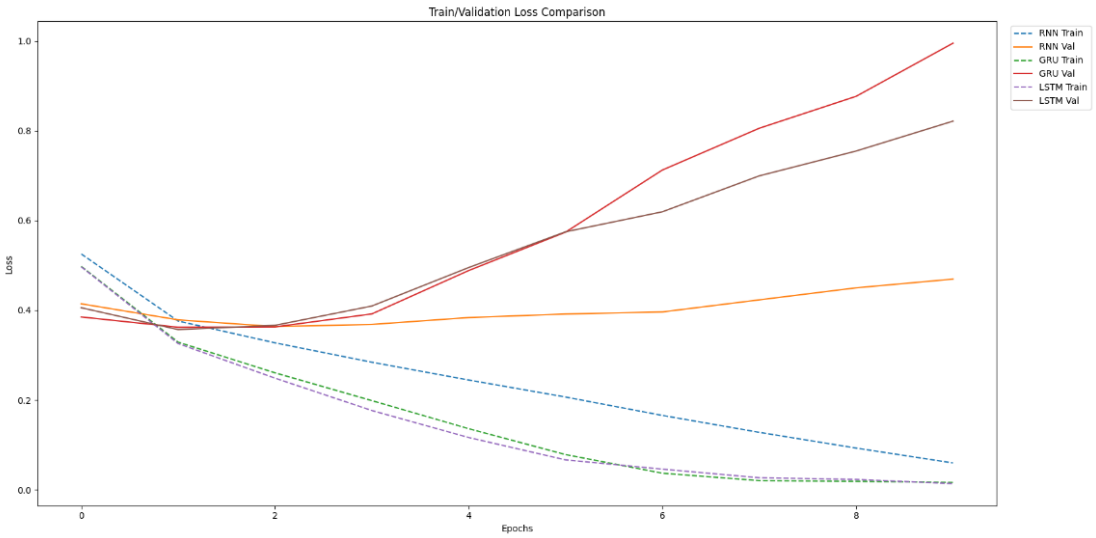
BiGRU-RNN

Η προεπεξεργασία των δεδομένων για το biGRU-RNN διαφέρει ελαφρά από πριν καθώς τώρα παριστάνουμε τις λέξεις με word embeddings. Για να το πετύχουμε αυτό χρησιμοποιούμε τον Text Vectorizer στον οποίο περνώντας σαν όρισμα το συνολικό μήκος του λεξιλογίου και το μήκος της κάθε ακολουθίας (το βρίσκουμε με την χρήση ευρετικής) που επιθυμούμε , συνδέουμε κάθε λέξη με έναν αριθμό φτιάχνοντας έτσι τις ενθέσεις λέξεων.

1. **Υλοποίηση**
   Για την υλοποίηση του biGRU-RNN έχουμε την κλάση **bigru_rnn**() η οποία δημιουργεί ένα RNN με ένα επίπεδο εισόδου το οποίο δέχεται μία μόνο κριτική σε μορφή συμβολοσειράς. Στην συνέχεια η χρήση του vectorizer μετατρέπει το κείμενο εισόδου σε ακέραιες τιμές και έπειτα υπάρχει το Embedding layer το οποίο μετατρέπει τις ακέραιες τιμές σε πυκνές διανυσματικές αναπαραστάσεις με διαστάσεις που καθορίζονται από το emb_size (=64). Μετά έχουμε τα Bidirectional layers τα οποία μας βοηθούν να επεξεργαζόμαστε τα δεδομένα και από τις δύο κατευθύνσεις. Χρησιμοποιούμε επίσης dropout layer (με πιθανότητα 0.5) για την αποφυγή του overfitting και στο τέλος έχουμε το επίπεδο εξόδου με 1 unit και activation function sigmoid ώστε να μας βγάλει την πιθανότητα της θετικής κατηγορίας.

2. **Αποτελέσματα (Loss over epochs, ROC, Metrics Table)**
   Εκπαιδεύουμε το RNN μας σε 10 εποχές, με batch_size 64 και χρησιμοποιούμε επίσης το 20% των training δεδομένων ως τα validation δεδομένα και προκύπτουν οι ακόλουθες καμπύλες εκ των οποίων η πρώτη είναι η μεταβολή του σφάλματος στα παραδείγματα εκπαίδευσης και επαλήθευσης καθώς προχωράνε τα epochs και η δεύτερη η ROC Curve.

```
RNN: Accuracy=0.8348, Precision=0.8357, Recall=0.8335, F1=0.8346
GRU: Accuracy=0.8421, Precision=0.8493, Recall=0.8317, F1=0.8404
LSTM: Accuracy=0.8394, Precision=0.8201, Recall=0.8696, F1=0.8441
```



3. **Συγκρίσεις με τις υλοποιήσεις μας**

biGRU-RNN VS Bernoulli Naive Bayes(from scratch)

Μπορούμε εύκολα να παρατηρήσουμε ότι το RNN μας πετυχαίνει χαμηλοτερες τιμές μετρικών και στα train και test δεδομένα (πλην του recall στα test) από την υλοποίηση μας για τον Bernoulli Naive Bayes.

```
RNN: Accuracy=0.8348, Precision=0.8357, Recall=0.8335, F1=0.8346
GRU: Accuracy=0.8421, Precision=0.8493, Recall=0.8317, F1=0.8404
LSTM: Accuracy=0.8394, Precision=0.8201, Recall=0.8696, F1=0.8441
```

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.88	0.85	0.86	0.81	0.87	0.83
10000	0.87	0.85	0.86	0.83	0.87	0.84
15000	0.86	0.85	0.85	0.82	0.86	0.84
20000	0.86	0.85	0.84	0.82	0.85	0.83
25000	0.85	0.85	0.85	0.83	0.85	0.84

biGRU-RNN VS Logistic Regression (from scratch)

Σύμφωνα με τον παρακάτω πίνακα τον RNN μας πετυχαίνει χειροτερα αποτελέσματα precision, recall και f1 σε συγκριση με τον αλγόριθμο του Logistic Regression.

```
RNN: Accuracy=0.8348, Precision=0.8357, Recall=0.8335, F1=0.8346
GRU: Accuracy=0.8421, Precision=0.8493, Recall=0.8317, F1=0.8404
LSTM: Accuracy=0.8394, Precision=0.8201, Recall=0.8696, F1=0.8441
```

	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
4000	1.00	0.83	1.00	0.84	1.00	0.84
8000	0.94	0.83	0.97	0.88	0.95	0.86
12000	0.94	0.87	0.93	0.85	0.94	0.86
16000	0.89	0.83	0.96	0.91	0.92	0.87
20000	0.90	0.85	0.94	0.90	0.92	0.87