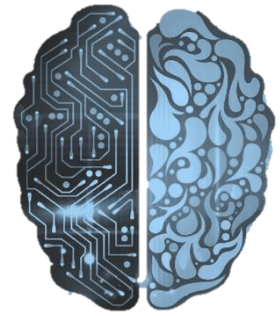


Artificial Intelligence

Machine Learning Project

Χρήστος Πατρινόπουλος 3200150

Αλβιόνα Μάντσο 3200098



Ο κώδικας είναι οργανωμένος ως εξής:

Imports:

Γίνονται όλες οι απαραίτητες εισαγωγές βιβλιοθηκών για την υλοποίηση όσων ακολουθούν.

Fetch Data:

Φορτώνονται τα δεδομένα από το imdb μέσω της συνάρτησης **data_fetch()**.

Vectorize Examples:

Χρησιμοποιείται η συνάρτηση **vectorize_examples(vocabulary, x_train, x_test=None)** για την μετατροπή των δεδομένα εκπαίδευσης και ελέγχου σε δυαδικά διανύσματα (εμφάνιση ή μη κάθε λέξης του vocabulary αντίστοιχα).

Create Vocabulary:

Δημιουργία του vocabulary (λεξικού) από τα δεδομένα εκπαίδευσης μέσω της συνάρτησης **create_vocabulary(x_train, y_train, n, k, m, l)**. Για να επιτευχθεί αυτό, αξιοποιούνται οι παράμετροι n, k, m όπως περιγράφονται καθώς και η βοηθητική συνάρτηση **IG(class_, feature)** (Info Gain) με τη βοήθεια της οποίας παίρνουμε τις l χρησιμοποιότερες λέξεις (υψηλότερο info gain) για τη δημιουργία του vocabulary.

Naïve Bayes:

Εδώ βρίσκεται η υλοποίηση της κλάσης του αφελούς ταξινομητή Bayes με πολυμεταβλητή μορφή Bernoulli (**class NaiveBayesClassifier()**). Υλοποιούνται οι συναρτήσεις **fit** και **predict**.

[Αναλυτικότερα παρακάτω.](#)

Logistic Regression:

Εδώ βρίσκεται η υλοποίηση της κλάσης της λογιστικής παλινδρόμησης (**class CustomLogisticRegression()**). Υλοποιούνται οι συναρτήσεις **fit** και **predict**, καθώς και μια βοηθητική σιγμοειδής (**pos_category_sigmoid(t)**) που είναι δηλωμένη ως static.

Επιπλέον υλοποιείται η συνάρτηση **find_best_regularizator(x_train, x_val_part, y_train_part, y_val_part)** για την εκτίμηση του βέλτιστου συντελεστή κανονικοποίησης λ , μέσω διαδοχικών πειραμάτων στα δεδομένα ανάπτυξης. [Αναλυτικότερα παρακάτω.](#)

Bidirectional GRU RNN:

Εδώ βρίσκεται η υλοποίηση της κλάσης του bidirectional RNN με GRU cells (**class BiGRU_RNN()**). Υλοποιούνται οι συναρτήσεις **fit** και **predict**. Επιπλέον χρησιμοποιείται η βοηθητική συνάρτηση **create_bi_GRU_RNN()** η οποία δημιουργεί το νευρωνικό δίκτυο. [Αναλυτικότερα παρακάτω.](#)

Curves:

Υλοποιούνται οι εξής συναρτήσεις:

- **learning_curve**(predictor, x_train, y_train, x_test, y_test, n_splits):
Δέχεται τα δεδομένα εκπαίδευσης και τα δεδομένα αξιολόγησης μαζί με τις αποκρίσεις τους, καθώς και τον ταξινομητή (predictor) προς αξιολόγηση.
Η συνάρτηση τεμαχίζει τα δεδομένα εκπαίδευσης σε n_splits μέρη και εκπαιδεύει επαναληπτικά τον ταξινομητή-predictor σε συνεχώς αυξανόμενο πλήθος δεδομένων, συσσωρεύοντας σταδιακά τα τεμάχια αυτά. Για κάθε μία από αυτές τις εκπαιδεύσεις (fits), προβλέπει (predicts) τις αποκρίσεις τόσο στα δεδομένα εκπαίδευσης όσο και στα δεδομένα αξιολόγησης, και συγκρίνοντας με τις σωστές αποκρίσεις (y_train, y_test) υπολογίζει 4 scores, *accuracy_score*, *precision_score*, *recall_score*, *f1_score*. Τα scores αυτά κρατούνται σε κατάλληλες λίστες για τη δημιουργία των διαγραμμάτων στο τέλος των επαναλήψεων. Επίσης εκμεταλλεύεται τα ήδη υπολογισμένα δεδομένα και παράγει συγκεντρωτικούς πίνακες με τα scores τους οποίους και επιστρέφει.
- **make_comparisons**(df_predictor1, df_predictor2):
Δέχεται τους συγκεντρωτικούς πίνακες των scores για 2 ταξινομητές προς σύγκριση (όπως οι πίνακες επιστράφηκαν από την συνάρτηση learning curve παραπάνω) και κατασκευάζει ένα heatmap που αναδεικνύει τις διαφορές των scores του δεύτερου ταξινομητή σε σχέση με τον πρώτο.
- **loss_plot**(history):
Χρησιμοποιείται για την παραγωγή των καμπυλών που δείχνουν τη μεταβολή του σφάλματος (loss) στα παραδείγματα εκπαίδευσης και ανάπτυξης, συναρτήσει του αριθμού των εποχών.

Training & Testing

Preparing Data:

Καλούμε τις συναρτήσεις που αναφέρθηκαν προηγουμένως και φορτώνουν τα δεδομένα, δημιουργούν το vocabulary και δημιουργούν τα δυαδικά διανύσματα των δεδομένων (παραδειγμάτων) εκπαίδευσης και αξιολόγησης.

Συγκεκριμένα ως προς τη δημιουργία του vocabulary χρησιμοποιούνται οι εξής παράμετροι:

- **n: 50**, Αφαιρούνται οι 50 πιο συχνές λέξεις. Το πλήθος επιλέχτηκε πειραματικά.
- **k: 85000**, Αφαιρούνται οι 85000 λιγότερο συχνές λέξεις. Το πλήθος επιλέχτηκε με παρατήρηση των συχνοτήτων εμφάνισης των λέξεων ύστερα από ταξινόμησή τους κατά αύξουσα συχνότητα εμφάνισης και αφαιρέθηκαν στην ουσία λέξεις μη χρήσιμες, συχνότητας ~1.
- **m: 2500**, Κρατούνται οι 2500 πιο συχνές λέξεις από αυτές που απομένουν. Το πλήθος επιλέχτηκε πειραματικά.
- **l: 1000**, Οι 2500 λέξεις που έμειναν παραπάνω είναι υποψήφιες επιλέξιμες και περνούν από τη συνάρτηση IG (Info Gain) μετά από την οποία καταλήγουν να παραμείνουν οι 1000 πιο χρήσιμες λέξεις. Το πλήθος επιλέχτηκε πειραματικά.



Γνώμονας σε όλες τις επιλογές ήταν η μείωση των υποψηφίων που περνούν από την συνάρτηση IG και τελικά η **μείωση των ιδιοτήτων και κατ' επέκταση των διαστάσεων των πινάκων** παραδειγμάτων που θα προκύψουν στα fit και predict.

Training, Using and Curves of Naïve Bayes Classifier:

Custom Naïve Bayes:

Εκπαίδευση (**fit**), πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για τον δικό μας ταξινομητή `NaiveBayesClassifier`.

Scikit-Learn Naïve Bayes:

Εκπαίδευση (**fit**), πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για την υλοποίηση του Sk-Learn `BernoulliNB`.

Comparisons:

Σύγκριση της δικής μας (custom) υλοποίησης με την υλοποίηση του Sk-Learn `BernoulliNB` με χρήση της συνάρτησης `make_comparisons`.

Training, Using and Curves of Logistic Regression Classifier:

Custom Logistic Regression:

Αρχικά βρίσκουμε και εμφανίζουμε τον βέλτιστο συντελεστή κανονικοποίησης λ καθώς και την ακρίβεια που πετυχαίνει στα δεδομένα ανάπτυξης (που είναι και εκείνα που αξιοποιούνται για την εύρεσή του). (χρήση της συνάρτησης `find_best_regularizator`).

Ακολουθεί, πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για τον δικό μας ταξινομητή `CustomLogisticRegression` με συντελεστή λ αυτόν που υπολογίστηκε.

Scikit-Learn Logistic Regression:

Εκπαίδευση (**fit**), πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για την υλοποίηση του Sk-Learn `LogisticRegression`.

Scikit-Learn SGDClassifier:

Εκπαίδευση (**fit**), πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για την υλοποίηση του Sk-Learn `SGDClassifier`.

Bidirectional RNN with GRU cells:

Προκαταρκτικά, υπολογίζουμε τον μέσο αριθμό λέξεων ενός κειμένου-παραδείγματος και με βάση τον αριθμό που προέκυψε, παίρνουμε ένα άνω όριο το οποίο χρειάζεται για την δημιουργία του **TextVectorization layer**. Στην συνέχεια δημιουργούμε το layer αυτό, το οποίο χρησιμοποιείται στην κατασκευή του νευρωνικού δικτύου.

Ακολουθεί εκπαίδευση (**fit**), πρόβλεψη (**predict**) και **αξιολόγηση** με `classification_report` και παραγωγή καμπυλών και πινάκων για το RNN μας.

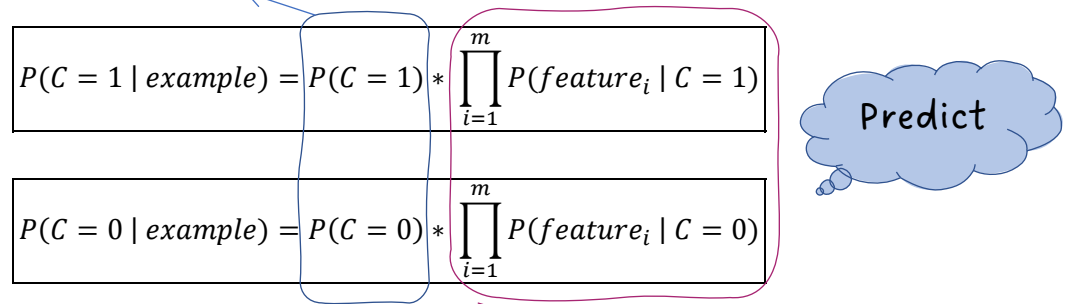
Naive Bayes Classifier

`fit(x_train_binary, y_train)`

Η μέθοδος **fit** χρησιμοποιείται για τον υπολογισμό και την αποθήκευση όλων των πιθανοτήτων που χρειάζονται κατά το **predict**:

- Υπολογίζονται οι πιθανότητες $P(C=1)$, $P(C=0)$ της θετικής και αρνητικής κατηγορίας αντίστοιχα.

$P(C = 1 example) = P(C = 1) * \prod_{i=1}^m P(feature_i C = 1)$
$P(C = 0 example) = P(C = 0) * \prod_{i=1}^m P(feature_i C = 0)$



- Στη συνέχεια υπολογίζονται οι δεσμευμένες πιθανότητες $P(feature_i = 1 | C=0)$, $P(feature_i = 1 | C=1)$ όπως χρειάζονται στον υπολογισμό του **γινομένου** κατά το **predict** (με υπόθεση δηλ. ανεξαρτησίας). Προφανώς δεν χρειάζεται να αποθηκεύσουμε τις πιθανότητες $P(feature_i = 0 | C=0) = 1 - P(feature_i = 1 | C=0)$, $P(feature_i = 0 | C=1) = 1 - P(feature_i = 1 | C=1)$, μειώνοντας τις απαιτήσεις μνήμης.

Στον παραπάνω υπολογισμό χρησιμοποιείται εξομάλυνση **Laplace** για να αποφευχθεί μηδενισμός ολόκληρης της πιθανότητας κατάταξης σε κατηγορία λόγω τοπικού μηδενισμού στο γινόμενο κατά το **predict**. Στον αριθμητή των παραπάνω πιθανοτήτων προστίθεται +1, ενώ στον παρονομαστή +2 (εφόσον έχουμε δύο δυνατές τιμές των features).

`predict(x_test_binary)`

Η μέθοδος **predict** χρησιμοποιείται για τον υπολογισμό της πιθανότητας κατάταξης σε κατηγορία για κάθε ένα από τα παραδείγματα προς αξιολόγηση που δίνονται.

- Για κάθε παράδειγμα (*example*) προς αξιολόγηση:

- Για κάθε feature του παραδείγματος:

- Υπολογίζονται οι πιθανότητες:

$P(C=0 | feature_i)$ πολλαπλασιάζοντας $P(C=0) * P(feature_i | C=0)$

$P(C=1 | feature_i)$ πολλαπλασιάζοντας $P(C=1) * P(feature_i | C=1)$

$$\begin{cases} P(feature_i = 1 | C=0), & \text{αν } feature_i = 1 \\ 1 - P(feature_i = 1 | C=0), & \text{αν } feature_i = 0 \end{cases}$$
$$\begin{cases} P(feature_i = 1 | C=1), & \text{αν } feature_i = 1 \\ 1 - P(feature_i = 1 | C=1), & \text{αν } feature_i = 0 \end{cases}$$

...οι οποίες πολλαπλασιάζονται σύμφωνα με τον τύπο, δίνοντας τις $P(C=0 | example)$ και $P(C=1 | example)$

Το παράδειγμα κατατάσσεται στην κατηγορία που έδωσε τη μεγαλύτερη από τις παραπάνω πιθανότητες.

Logistic Regression

fit(x_train_binary, y_train)

- Αρχικά χωρίζονται τα παραδείγματα σε παραδείγματα εκπαίδευσης και επικύρωσης, με ποσοστό επικύρωσης 20%.
- Τα βάρη όλων των ιδιοτήτων αρχικοποιούνται στο 0.

Για έναν μέγιστο αριθμό επαναλήψεων/εποχών (n_iters), αναδιατάσσουμε τυχαία τα παραδείγματα σε κάθε επανάληψη (ώστε τα βήματα προς τη σύγκλιση των βαρών να είναι ανεξάρτητα από επανάληψη σε επανάληψη) και για κάθε παράδειγμα ενημερώνουμε τα βάρη (συμπεριλαμβανομένου του bias factor) σύμφωνα με τον τύπο:

$$\vec{w} = (1 - 2 * \lambda * \eta) \vec{w} + \eta * \sum_{i=1}^m [y_i - P(C_+ | \vec{x}(i))] * \vec{x}(i)$$

Συνεπώς εφόσον η ενημέρωση των βαρών γίνεται με βάση ένα παράδειγμα κάθε φορά, το $\vec{x}(i)$, έχουμε **στοχαστική** ανάβαση κλίσης.

Για τον τερματισμό των επαναλήψεων χωρίς να έχουν ξεπεραστεί οι μέγιστες, υπολογίζεται με **Early Stopping** το accuracy score στα δεδομένα επικύρωσης. Αν αυτό δεν βελτιωθεί για 20 το πολύ επαναλήψεις σταματάει το fit και κρατούνται τα καλύτερα βάρη που είχαν βρεθεί.

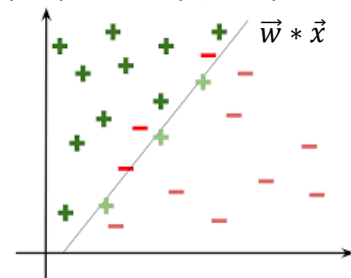
predict(x_test_binary)

■ Για κάθε παράδειγμα προς αξιολόγηση:

■ Υπολογίζεται το γινόμενο του διανύσματος των βαρών (όπως μαθεύτηκαν από το fit) με το **διάνυσμα ιδιοτήτων** του τρέχοντος παραδείγματος ($\vec{w} * \vec{x}$). Η κατάταξη σε κατηγορία γίνεται σύμφωνα με το πρόσημο του γινομένου αυτού:

- Κατάταξη στην θετική κατηγορία (1), αν είναι θετικό (+)
- Κατάταξη στην αρνητική κατηγορία (0), αν είναι αρνητικό (-)

$$(\vec{w} * \vec{x}) \begin{cases} \rightarrow pos \rightarrow C=1 \\ \rightarrow neg \rightarrow C=0 \end{cases}$$

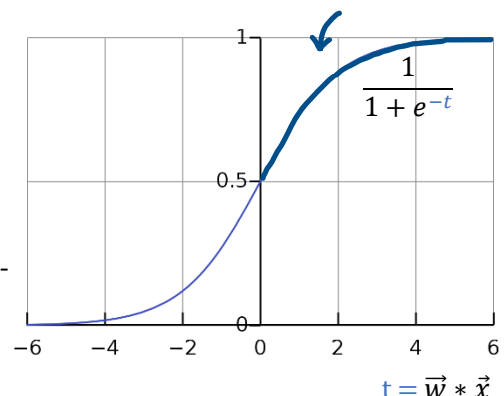


Σημειώνουμε ότι η απόφαση κατάταξης σύμφωνα με το πρόσημο του γινομένου ($\vec{w} * \vec{x}$) έχει ακριβώς όμοια αποτελέσματα με μια ελαφρώς διαφορετική συχνά χρησιμοποιούμενη προσέγγιση, σύμφωνα με την οποία η απόφαση κατάταξης λαμβάνεται από την τιμή του:

$$\text{sigmoid}(\vec{w} * \vec{x}) \begin{cases} \geq 0.5 \rightarrow C=1 \\ < 0.5 \rightarrow C=0 \end{cases}$$

... Αυτό γιατί $\text{sigmoid}(\vec{w} * \vec{x}) \geq 0.5 \Leftrightarrow \vec{w} * \vec{x} \geq 0$

Ωστόσο με τον προηγούμενο τρόπο κατάταξης μειώνονται οι απαιτούμενες πράξεις εφόσον δεν χρησιμοποιείται η σιγμοειδής συνάρτηση.



λ : regularization factor

Για την εύρεση του **βέλτιστου συντελεστή κανονικοποίησης** χρησιμοποιείται η συνάρτηση **find_best_regularizer**, στην οποία γίνονται επαναληπτικές (μέσω ενός εύρους τιμών του λ : $1e-15$ έως $0.99 + 1e-15$) αξιολογήσεις (accuracy score) πάνω στα δεδομένα επικύρωσης και επιστρέφονται πληροφορίες που έχουν να κάνουν με το καλύτερο λ , καθώς και ο ίδιος ο ταξινομητής που είχε το λ αυτό. Εάν το accuracy score δε βελτιωθεί εντός 5 διαδοχικών επαναλήψεων/δοκιμών τότε τερματίζεται η αναζήτηση του λ .

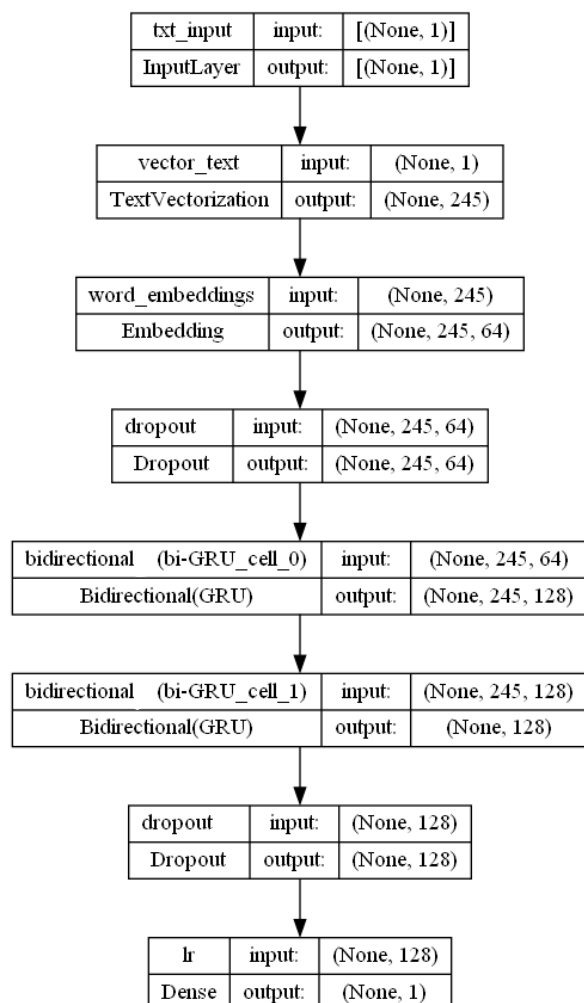
Το βέλτιστο λ που επιτυγχάνεται είναι το μικρότερο του παραπάνω εύρους ($1e-15$), ενώ μάλιστα μετά από πείραμα στο οποίο το λ τέθηκε 0 τα αποτελέσματα ήταν ακόμη καλύτερα (δηλ. χωρίς την χρήση κανονικοποίησης). Αυτό είναι λογικό, αφού, όπως θα φανεί και στα διαγράμματα παρακάτω, δεν υπάρχει πρόβλημα high variance (overfitting), εφόσον οι καμπύλες μάθησης των δεδομένων εκπαίδευσης και αξιολόγησης έχουν συγκλίνει. Συνεπώς η όποια προσπάθεια βελτίωσης θα κινούνταν μάλλον προς μείωση του λ .

η : learning rate

Χρησιμοποιείται η τιμή 0.001 που είναι μια τυπική τιμή συχνά προτιμώμενη για αυτό το σκοπό.

Max iterations

Ορίζεται η τιμή 100, καθώς πειραματικά παρατηρήθηκε ότι με χρήση του early stopping οι επαναλήψεις/εποχές που γίνονταν ήταν γενικά λιγότερες από 100.



RNN

Τα επίπεδα του RNN που χρησιμοποιείται και οι διαστάσεις τους φαίνονται στη διπλανή εικόνα. Επίσης γίνεται αντιληπτό ότι πρόκειται για **Bidirectional GRU cell RNN με 2 biGRU layers**.

Fit: Με κάθε κλήση της fit δημιουργείται εκ νέου το νευρωνικό δίκτυο (κλήση της βοηθητικής create_bi_GRU_RNN()) και γίνεται compile. Στη συνέχεια πάνω σε αυτό καλείται η τυποποιημένη fit (του keras model) για να εκπαιδευτεί το νευρωνικό. Σκοπός είναι να μην διατηρούνται τα δεδομένα προηγούμενων κλήσεων της fit, δηλ να «καθαρίζεται» η μνήμη του νευρωνικού.

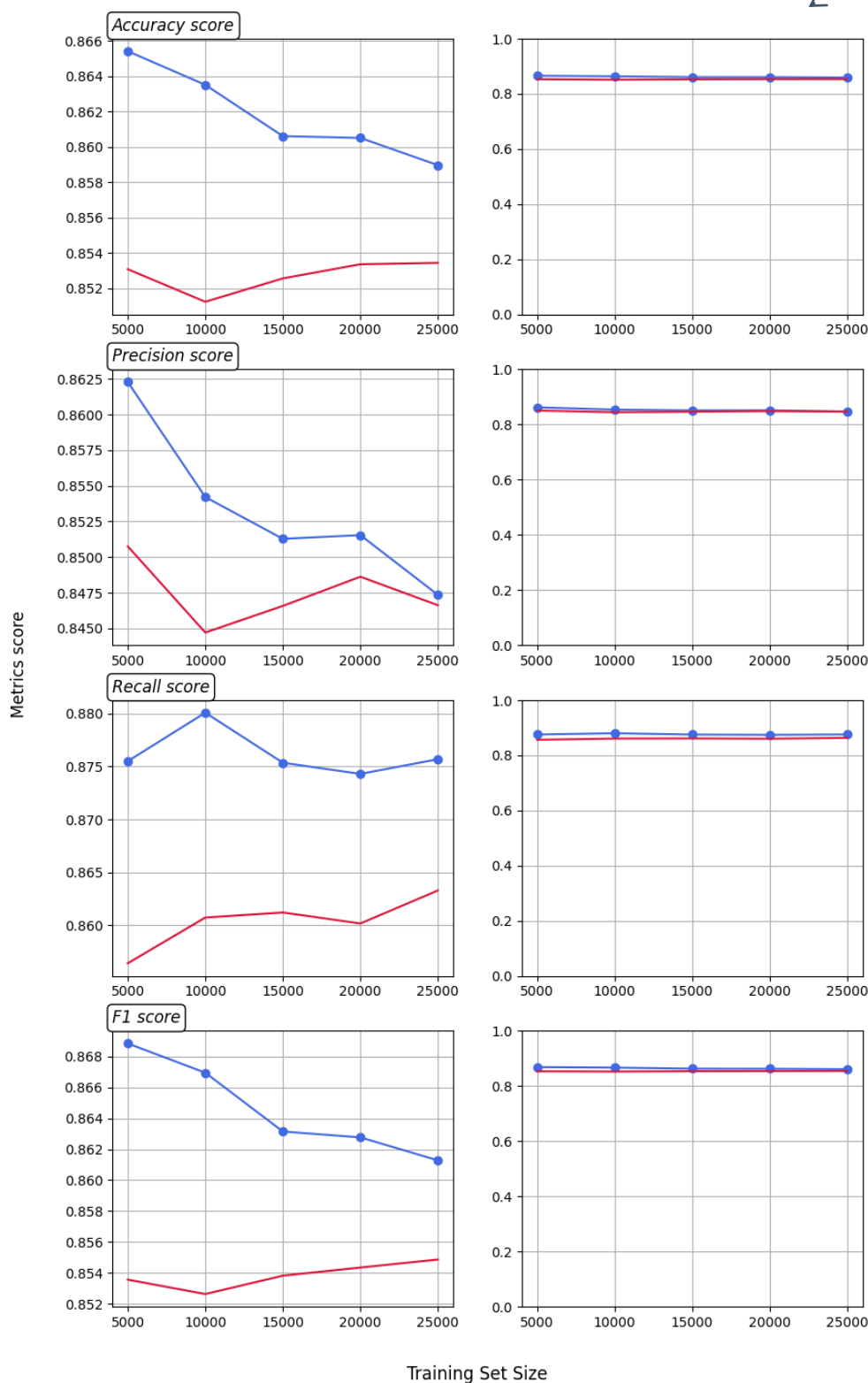
Predict: Η predict καλεί την αντίστοιχη τυποποιημένη predict (του keras model). Στη συνέχεια οι προβλέψεις (predictions) μετατρέπονται από πιθανοτικές σε δυαδικές. Σκοπός είναι να επαναχρησιμοποιηθεί για τα διαγράμματά μας η συνάρτηση learning_curves.

Σημειώνουμε επιπλέον ότι η παράμετρος SEQ_MAX_LENGTH που δίνεται στο TextVectorization layer υπολογίζεται ως το μέσο μήκος (πλήθος λέξεων) των κειμένων και για καλύτερα στατιστικά αποτελέσματα λαμβάνεται το άνω φράγμα του 99% διαστήματος εμπιστοσύνης (242) προσαυξανόμενο κατά ένα περιθώριο 3 λέξεων ($242+3=245$).

	Accuracy Train	Accuracy Test	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
Train Size								
5000	0.865	0.853	0.862	0.851	0.875	0.856	0.869	0.854
10000	0.864	0.851	0.854	0.845	0.880	0.861	0.867	0.853
15000	0.861	0.853	0.851	0.847	0.875	0.861	0.863	0.854
20000	0.861	0.853	0.852	0.849	0.874	0.860	0.863	0.854
25000	0.859	0.853	0.847	0.847	0.876	0.863	0.861	0.855

Custom Naive Bayes

— Training
— Testing



Παρατηρούμε ότι οι καμπύλες ορθότητας (**accuracy score**) και **f1 score** ακολουθούν την αναμενόμενη μορφή κατά την οποία τα ποσοστά στα δεδομένα εκπαίδευσης μειώνονται ενώ στα δεδομένα αξιολόγησης αυξάνονται καθώς ο ταξινομητής εκπαιδεύεται σε ολοένα και περισσότερα δεδομένα. Αυτό σημαίνει ότι όσο περισσότερο εκπαιδεύεται ο ταξινομητής, τόσο πιο δύσκολο γίνεται να αποστηθίσει (και άρα να προβλέπει σωστά) τα δεδομένα εκπαίδευσης, ενώ παράλληλα αυξάνεται η ικανότητά του για γενίκευση.

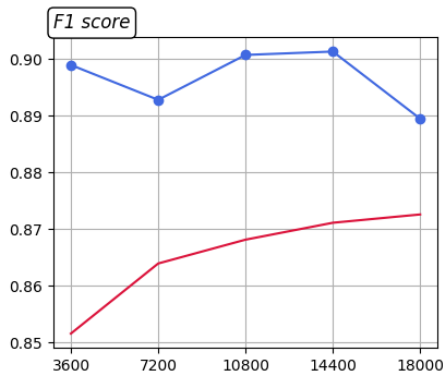
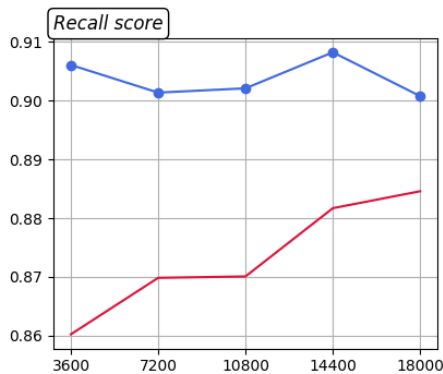
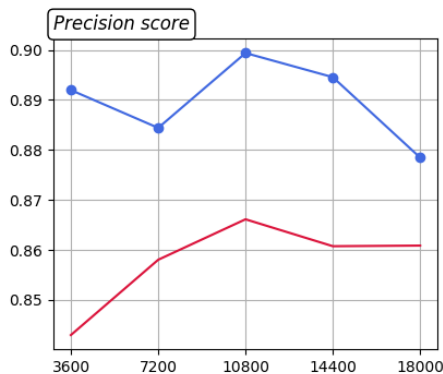
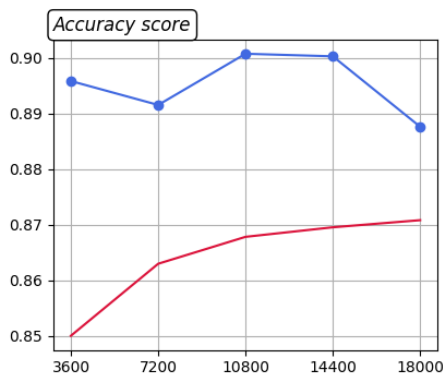
Τελικά παρατηρούμε ότι οι δύο καμπύλες **συγκλίνουν** και μάλιστα σε ένα ικανοποιητικό ποσοστό ορθότητας, επομένως δεν υπάρχουν περιθώρια βελτίωσης.

	Accuracy Train	Accuracy Test	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
Train Size								
3600	0.896	0.850	0.892	0.843	0.906	0.860	0.899	0.852
7200	0.892	0.863	0.884	0.858	0.901	0.870	0.893	0.864
10800	0.901	0.868	0.899	0.866	0.902	0.870	0.901	0.868
14400	0.900	0.870	0.895	0.861	0.908	0.882	0.901	0.871
18000	0.888	0.871	0.878	0.861	0.901	0.885	0.889	0.873

Custom Logistic Regression

—●— Training
— Testing

Metrics score



Training Set Size

(Υπερ)παράμετροι

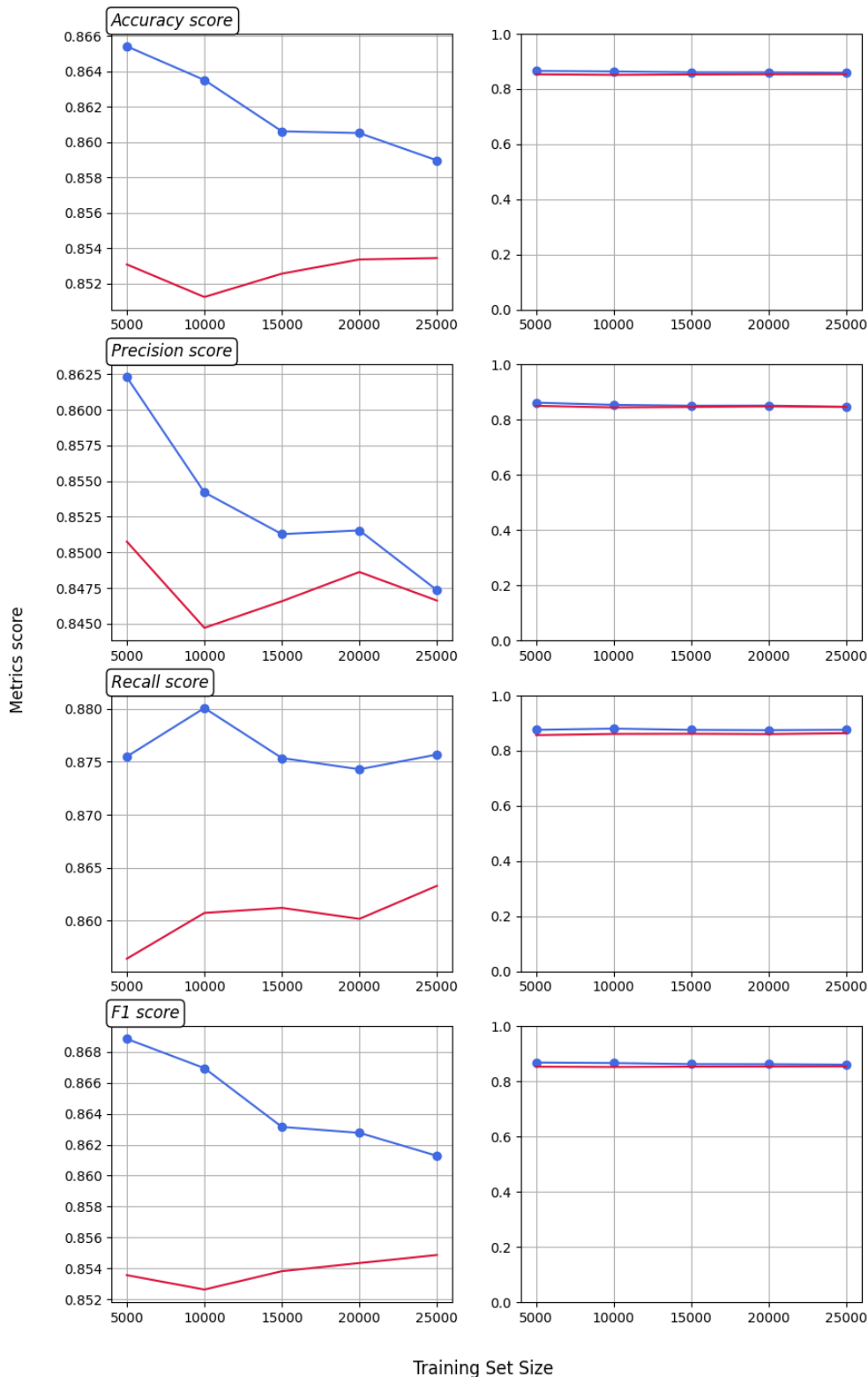
Learning rate (η): 0.001
Regularization factor (λ): 1e-15
Max iterations: 100
Χρήση Early Stopping

Παρατηρούμε και εδώ ότι η ορθότητα (**accuracy score**) και το **f1 score** μειώνονται ενώ στα δεδομένα αξιολόγησης αυξάνονται καθώς ο ταξινομητής εκπαιδεύεται σε ολοένα και περισσότερα δεδομένα. Αυτό σημαίνει ότι όσο περισσότερο εκπαιδεύεται ο ταξινομητής, τόσο πιο δύσκολο γίνεται να αποστηθίσει (και άρα να προβλέπει σωστά) τα δεδομένα εκπαίδευσης, ενώ παράλληλα αυξάνεται η ικανότητά του για γενίκευση.

Τελικά παρατηρούμε ότι οι δύο καμπύλες **συγκλίνουν** και μάλιστα σε ένα ικανοποιητικό ποσοστό ορθότητας, επομένως δεν υπάρχουν περιθώρια βελτίωσης.

	Accuracy Train	Accuracy Test	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
Train Size								
5000	0.865	0.853	0.862	0.851	0.875	0.856	0.869	0.854
10000	0.864	0.851	0.854	0.845	0.880	0.861	0.867	0.853
15000	0.861	0.853	0.851	0.847	0.875	0.861	0.863	0.854
20000	0.861	0.853	0.852	0.849	0.874	0.860	0.863	0.854
25000	0.859	0.853	0.847	0.847	0.876	0.863	0.861	0.855

BernoulliNB



Συγκριτικά με την δική μας υλοποίηση παρατηρούμε ακριβώς **όμοια** συμπεριφορά και αποτελέσματα σε όλες τις μετρικές. Αυτό είναι λογικό καθώς δεν υπάρχει παράγοντας τυχαιότητας στον αλγόριθμο.

	Accuracy Train	Accuracy Test	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
Train Size								
5000	0.915	0.829	0.921	0.838	0.912	0.816	0.916	0.827
10000	0.891	0.847	0.898	0.857	0.885	0.833	0.891	0.845
15000	0.886	0.852	0.913	0.880	0.855	0.815	0.883	0.846
20000	0.890	0.862	0.907	0.876	0.870	0.842	0.888	0.859
25000	0.888	0.867	0.864	0.842	0.922	0.903	0.892	0.871

SGDClassifier

— Training
— Testing

(Υπερ)παράμετροι:

Loss: 'log'

Max iterations: 100

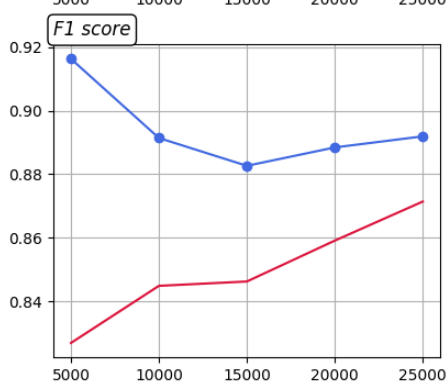
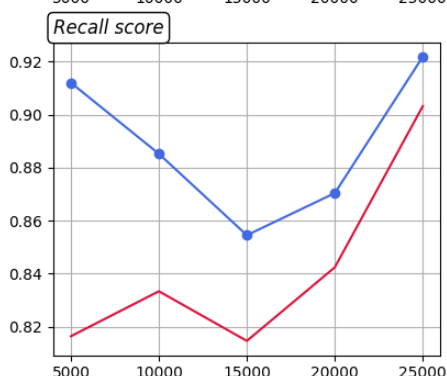
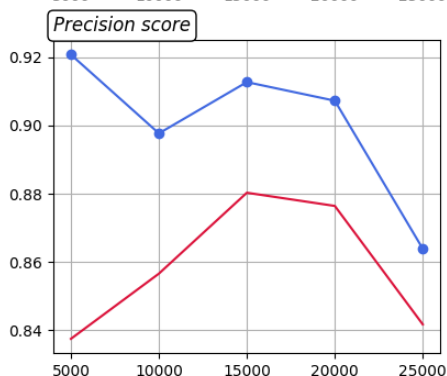
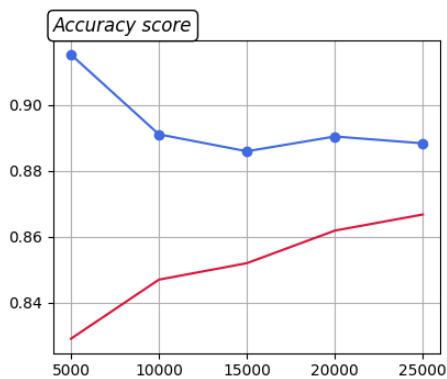
Χρήση Early Stopping

Συγκριτικά με την δική μας υλοποίηση παρατηρούμε μια παρόμοια συμπεριφορά στην ορθότητα (**accuracy score**) και στο **f1 score** ενώ και οι τελικές τιμές όλων των μετρικών βρίσκονται πολύ κοντά στις δύο υλοποιήσεις.

Είναι ωστόσο λογικό να υπάρχουν αποκλίσεις, καθώς:

- 1) Υπάρχει τυχαιότητα στους αλγόριθμους (π.χ. αναδιάταξη - shuffle - παραδειγμάτων σε κάθε επανάληψη στο fit),
- 2) Ο δικός μας αλγόριθμος χρησιμοποιεί στοχαστική ανάβαση κλίσης ενώ ο SGD (Stochastic Gradient Descent) χρησιμοποιεί κατάβαση,
- 3) Παρότι κάποιες παράμετροι ρυθμίστηκαν έτσι ώστε να συμφωνούν με τη δική μας υλοποίηση, δεν έχουν ρυθμιστεί όλες (π.χ. το learning rate).

Metrics score



Training Set Size

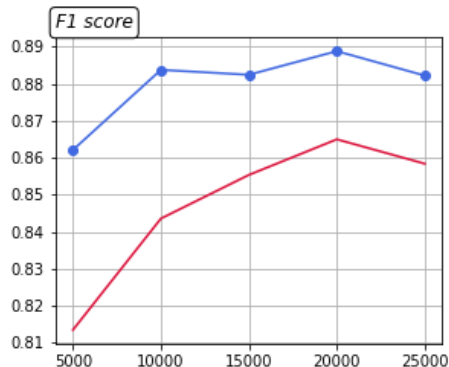
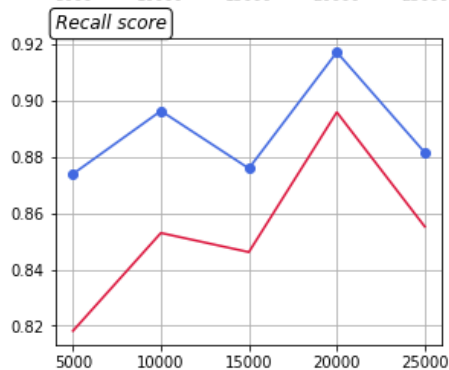
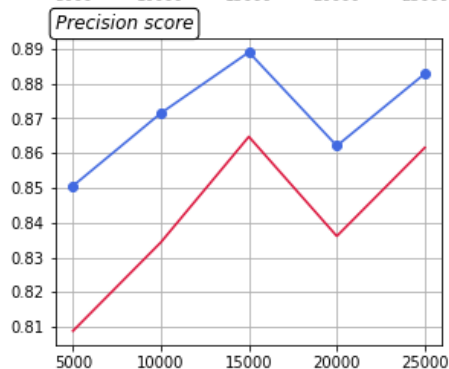
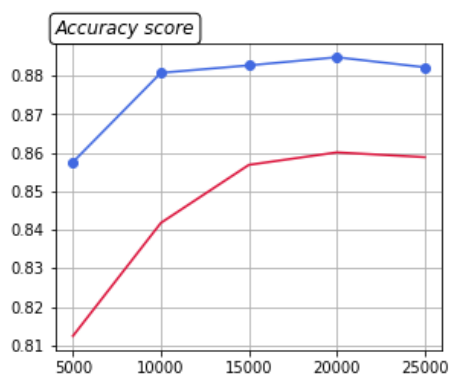
Στο αρχείο του κώδικα γίνεται επιπλέον σύγκριση με τον αλγόριθμο **LogisticRegression** η οποία για λόγους συντομίας δεν παρουσιάζεται εδώ. Σημειώνουμε ότι ισχύουν εν γένει τα παραπάνω, έχοντας και σε αυτήν την περίπτωση μικρές αποκλίσεις από τη δική μας υλοποίηση.

	Accuracy Train	Accuracy Test	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
Train Size								
5000	0.858	0.812	0.851	0.809	0.874	0.818	0.862	0.813
10000	0.881	0.842	0.871	0.834	0.896	0.853	0.884	0.844
15000	0.883	0.857	0.889	0.865	0.876	0.846	0.882	0.855
20000	0.885	0.860	0.862	0.836	0.917	0.896	0.889	0.865
25000	0.882	0.859	0.883	0.862	0.882	0.855	0.882	0.858

BiGRU RNN

—●— Training
— Testing

Metrics score



Training Set Size

(Υπερ)παράμετροι:

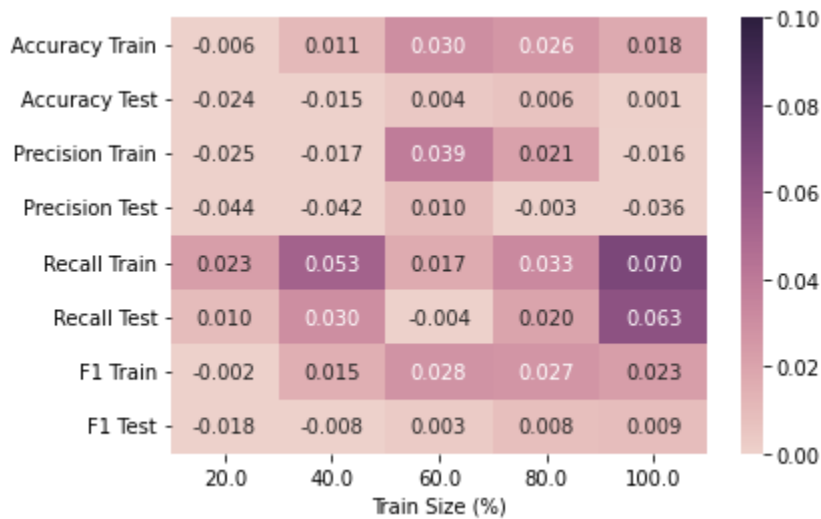
biGRU RNN layers: 2
epochs: 3
emb_size: 64
h_size: 64
batch_size: 32

Παρατηρούμε ότι η ορθότητα (**accuracy score**) είναι αυξανόμενη, όπως και το πιο αντιπροσωπευτικό **f1 score**. Αυτό σημαίνει ότι όσο περισσότερο εκπαιδεύεται ο ταξινομητής, τόσο καλύτερη γίνεται η επίδοσή του.

Τελικά παρατηρούμε ότι οι δύο καμπύλες **συγκλίνουν** και μάλιστα σε ένα ικανοποιητικό ποσοστό ορθότητας, αλλά ίσως να υπάρχουν μικρά περιθώρια βελτίωσης για περισσότερες εποχές κατά την εκπαίδευση.

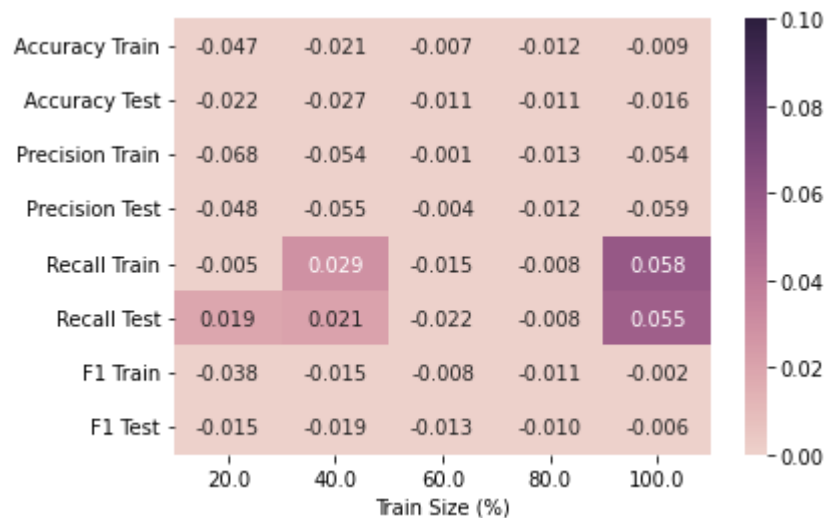
Συγκριτικά με τους δύο προηγούμενους αλγόριθμους, παρατηρούμε βελτίωση στα τελικά ποσοστά σε σχέση με τον Naive Bayes (~2% σε γενικές γραμμές) και μικρές διαφοροποιήσεις σε σχέση με τον Logistic Regression.

Οπτικές συγκρίσεις



Δική μας υλοποίηση NaiveBayesClassifier – BiGRU_RNN

Στο RNN παρατηρείται μείωση γενικά των μετρικών πέρα από το recall το οποίο αυξάνεται σημαντικά. Η ισόποση σχεδόν μείωση του precision οδηγεί σε παρόμοιο f1 score (ελαφρώς χαμηλότερο στο RNN).



Δική μας υλοποίηση CustomLogisticRegression – BiGRU_RNN

(Υπερ)παράμετροι:

biGRU RNN layers: 2
epochs: 10
emb_size: 64
h_size: 64
batch_size: 32
validation_split: 0.2

Παρατηρούμε ότι καθώς αυξάνεται ο αριθμός των εποχών το σφάλμα (loss) στα δεδομένα **εκπαίδευσης** μειώνεται συνεχώς γιατί το δίκτυο μαθαίνει σε κάθε εποχή όλο και περισσότερο τα παραδείγματα εκπαίδευσης, ώστε τα πηγαίνει πολύ καλά σε αυτά.

Από την άλλη, στα δεδομένα **επικύρωσης** το σφάλμα μειώνεται μέχρι κάποια εποχή (περίπου στην 6^η), ενώ στη συνέχεια περαιτέρω αύξηση των εποχών οδηγεί σε αύξηση του σφάλματος (χωρίς να αποκλείεται αυτή η τάση να αλλάξει στην πορεία σε επόμενες εποχές).

