

## Μέρος Α' :

### Naïve Bayes :

Στο αρχείο κώδικα **NaiveBayes.py** :

Αρχικά χρησιμοποιούμε την συνάρτηση **imdb.load\_data** από την Keras για να φορτώσει το σύνολο δεδομένων του IMDB. Το σύνολο δεδομένων χωρίζεται σε σύνολα εκπαίδευσης και ελέγχου, που αντιπροσωπεύονται από (X\_train, y\_train) και (X\_test, y\_test) .

Το **X\_train** και το **X\_test** είναι λίστες ακολουθιών, όπου κάθε μια αντιπροσωπεύει μια κριτική ταινίας ως μια ακολουθία δεικτών λέξεων. Αυτοί οι δείκτες αντιστοιχούν στις 1000 πιο συχνές λέξεις στο σύνολο δεδομένων (num\_words=1000). Το **y\_train** και το **y\_test** είναι λίστες ετικετών (0 για αρνητικές κριτικές, 1 για θετικές κριτικές).

Ορίζουμε τις υπερπαραμέτρους σύμφωνα με την εκφώνηση :

- **m=500**
- **n=0.9** (Λέξεις που εμφανίζονται σε περισσότερο από το 90% )
- **k=50**

### class CustomNaiveBayes:

Δημιουργούμε την κλάση για υλοποίηση του αλγορίθμου .

Η κλάση διατηρεί χαρακτηριστικά για τις ετικέτες κλάσεων (classes), τις πιθανότητες κλάσεων (class\_probs) και τις πιθανότητες χαρακτηριστικών που δίνονται σε κάθε κλάση (feature\_probs). Υπολογίζονται με την μέθοδο **fit** .

Η μέθοδος **predict** δίνει ένα πίνακα NumPy (np.array(predictions)) που περιέχει τις προβλεπόμενες ετικέτες κλάσεων για κάθε δείγμα εισόδου.

### Εκπαίδευση Custom Naive Bayes Classifier:

Δημιουργείται ένα instance της κλάσης CustomNaiveBayes (custom\_nb\_classifier).

Ο ταξινομητής εκπαιδεύεται στα δεδομένα εκπαίδευσης χρησιμοποιώντας τη μέθοδο fit.

### Πρόβλεψη για σετ εκπαίδευσης και ελέγχου:

Οι προβλέψεις γίνονται τόσο στο σετ εκπαίδευσης (X\_train\_binary) όσο και στο σετ ελέγχου (X\_test\_binary) και αποθηκεύονται στα (y\_train\_dev\_pred\_custom\_nb) και (y\_test\_dev\_pred\_custom\_nb) αντίστοιχα.

### Αξιολόγηση απόδοσης:

Υπολογίζεται και τυπώνεται ο πίνακας με το ποσοστό ορθότητας (accuracy) καθώς και τα precision, recall, F1-score για τα δεδομένα ελέγχου και εκπαίδευσης .

### Δημιουργία διαγραμμάτων Καμπυλών :

Οι λίστες (train\_accuracies\_custom\_nb, test\_accuracies\_custom\_nb, precisions\_custom\_nb, recalls\_custom\_nb, f1\_scores\_custom\_nb) αρχικοποιούνται για την αποθήκευση τιμών.

Το sizes είναι μια λίστα που περιέχει διαφορετικά μεγέθη υποσυνόλων.

Ένας βρόχος (for) επαναλαμβάνεται σε διαφορετικά μεγέθη υποσυνόλου.

Για κάθε επανάληψη, δημιουργείται νέο instance του ταξινομητή CustomNaiveBayes και εκπαιδεύεται σε ένα υποσύνολο των δεδομένων εκπαίδευσης (X\_train\_binary[:size].toarray() και y\_train[:size]).

Στη συνέχεια γίνονται προβλέψεις τόσο στο υποσύνολο εκπαίδευσης όσο και στο υποσύνολο ελέγχου. Υπολογίζονται μετρήσεις accuracy, precision, recall, f1 και αποθηκεύονται στις αντίστοιχες λίστες.

Τα αντίστοιχα διαγράμματα εμφανίζονται χρησιμοποιώντας το plt.show().

### RandomForest :

Στο αρχείο κώδικα **RandomForest.py** :

Αρχικά χρησιμοποιούμε την συνάρτηση **imdb.load\_data** από την Keras για να φορτώσει το σύνολο δεδομένων του IMDB. Το σύνολο δεδομένων χωρίζεται σε σύνολα εκπαίδευσης και ελέγχου, που αντιπροσωπεύονται από (X\_train, y\_train) και (X\_test, y\_test) .

Το **X\_train** και το **X\_test** είναι λίστες ακολουθιών, όπου κάθε μια αντιπροσωπεύει μια κριτική ταινίας ως μια ακολουθία δεικτών λέξεων. Αυτοί οι δείκτες αντιστοιχούν στις 1000 πιο συχνές λέξεις στο σύνολο δεδομένων (num\_words=1000). Το **y\_train** και το **y\_test** είναι λίστες ετικετών (0 για αρνητικές κριτικές, 1 για θετικές κριτικές).

Ορίζουμε τις υπερπαραμέτρους σύμφωνα με την εκφώνηση :

- **m=1000**
- **n=0.9** (Λέξεις που εμφανίζονται σε περισσότερο από το 90% )
- **k=50**
- **n\_estimators = 10**
- **max\_depth = 3**

### class myRandomForestClassifier:

Υλοποίηση Random Forest

Ο αλγόριθμος Random Forest περιλαμβάνει τη δημιουργία πολλών δέντρων απόφασης, τα οποία εκπαιδεύονται σε διάφορα υποσύνολα του συνόλου εκπαίδευσης. Τα δέντρα αυτά λειτουργούν ως "συντηρητές γνώσης" και παρέχουν προβλέψεις. Η τελική πρόβλεψη γίνεται με ψηφοφορία ή τον μέσο όρο των προβλέψεων όλων των δέντρων.

Η υλοποίηση του Random Forest γίνεται με την κλάση `myRandomForestClassifier`, η οποία περιλαμβάνει τις μεθόδους `fit` (για την εκπαίδευση) και `predict` (για τις προβλέψεις).

Το Random Forest αποτελείται από πολλά δέντρα ID3 Decision Tree που εκπαιδεύονται ανεξάρτητα.

#### **class ID3DecisionTree :**

Αυτή η κλάση αρχικοποιεί ένα δέντρο ID3 και περιλαμβάνει τις μεθόδους `fit` (για την εκπαίδευση), `_fit` (αναδρομική κατασκευή του δέντρου), `_find_best_split` (εύρεση του καλύτερου διαχωρισμού με βάση το Information Gain), `calculate_ig` (υπολογισμός Information Gain) και `predict` (για τις προβλέψεις).

Κατά την εκπαίδευση, ο αλγόριθμος εξερευνά αναδρομικά τα δεδομένα εκπαίδευσης και κατασκευάζει το δέντρο απόφασης. Αυτό γίνεται μέσω της ιδιωτικής μεθόδου `_fit`, η οποία δέχεται ως είσοδο τα δεδομένα εκπαίδευσης και τις ετικέτες τους, καθώς και το βάθος του τρέχοντος κόμβου. Η `_fit` αναδρομικά εξετάζει όλες τις δυνατές διαιρέσεις των δεδομένων και επιλέγει τη βέλτιστη διαχωριστική συνθήκη για κάθε κόμβο, βασιζόμενη στο κέρδος πληροφορίας (IG). Αν δεν μπορεί να βρει μια καλή διαχωριστική συνθήκη ή έχει φτάσει το μέγιστο βάθος, δημιουργεί ένα φύλλο και τερματίζει την αναδρομική διαδικασία. Η διαδικασία αυτή συνεχίζεται για κάθε υποσύνολο δεδομένων που προκύπτει από τη διαχωριστική συνθήκη.

Δέντρο ID3 Decision Tree : Η βασική μονάδα του Random Forest είναι το δέντρο απόφασης, το οποίο χρησιμοποιεί τον αλγόριθμο ID3 για τη δημιουργία. Κάθε δέντρο απόφασης δημιουργείται αναδρομικά και χωρίζει το σύνολο δεδομένων σε υποσύνολα με βάση το χαρακτηριστικό που προσφέρει τον καλύτερο διαχωρισμό, βασιζόμενο στον υπολογισμό του Κέρδους Πληροφορίας (Information Gain).

Η υλοποίηση του ID3 Decision Tree περιλαμβάνει τα εξής βασικά στοιχεία:

Κόμβος (Node): Ορίζεται ένας κόμβος που αντιπροσωπεύει τη δομή ενός δέντρου. Κάθε κόμβος έχει ένα δείκτη χαρακτηριστικού, ένα κατώτατο όριο για διαίρεση, μια ετικέτα κλάσης για φύλλο, και αριστερό και δεξί υποδέντρο.

#### **Εκπαίδευση Custom Random Forest Classifier:**

Δημιουργείται ένα instance της κλάσης `myRandomForestClassifier` (`n_estimators`, `max_depth`)

Ο ταξινομητής εκπαιδεύεται στα δεδομένα εκπαίδευσης χρησιμοποιώντας τη μέθοδο `fit`.

#### **Πρόβλεψη για σετ εκπαίδευσης και ελέγχου:**

Οι προβλέψεις γίνονται τόσο στο σετ εκπαίδευσης (`X_train_binary`) όσο και στο σετ ελέγχου (`X_test_binary`) και αποθηκεύονται στα (`y_train_pred_rf`) και (`y_test_pred_rf`) αντίστοιχα.

#### **Αξιολόγηση απόδοσης:**

Υπολογίζεται και τυπώνεται ο πίνακας με το ποσοστό ορθότητας (accuracy) καθώς και τα precision, recall, F1-score για τα δεδομένα ελέγχου και εκπαίδευσης .

#### Δημιουργία διαγραμμάτων Καμπυλών :

Οι λίστες (train\_accuracies, test\_accuracies, precisions, recalls, f1\_scores) αρχικοποιούνται για την αποθήκευση τιμών.

Το sizes είναι μια λίστα που περιέχει διαφορετικά μεγέθη υποσυνόλων.

Ένας βρόχος (for) επαναλαμβάνεται σε διαφορετικά μεγέθη υποσυνόλου.

Για κάθε επανάληψη, δημιουργείται νέο instance του ταξινομητή `myRandomForestClassifier` και εκπαιδεύεται σε ένα υποσύνολο των δεδομένων εκπαίδευσης (`X_train_binary[:size].toarray()` και `y_train[:size]`).

Στη συνέχεια γίνονται προβλέψεις τόσο στο υποσύνολο εκπαίδευσης όσο και στο υποσύνολο ελέγχου. Υπολογίζονται μετρήσεις accuracy, precision, recall, f1 και αποθηκεύονται στις αντίστοιχες λίστες.

Τα αντίστοιχα διαγράμματα εμφανίζονται χρησιμοποιώντας το `plt.show()`.

### Logistic Regression:

Στο αρχείο κώδικα **LogisticRegression.py** :

Αρχικά χρησιμοποιούμε την συνάρτηση `imdb.load_data` από την Keras για να φορτώσει το σύνολο δεδομένων του IMDB. Το σύνολο δεδομένων χωρίζεται σε σύνολα εκπαίδευσης και ελέγχου, που αντιπροσωπεύονται από (`X_train`, `y_train`) και (`X_test`, `y_test`) .

Το **X\_train** και το **X\_test** είναι λίστες ακολουθιών, όπου κάθε μια αντιπροσωπεύει μια κριτική ταινίας ως μια ακολουθία δεικτών λέξεων. Αυτοί οι δείκτες αντιστοιχούν στις 1000 πιο συχνές λέξεις στο σύνολο δεδομένων (`num_words=1000`). Το **y\_train** και το **y\_test** είναι λίστες ετικετών (0 για αρνητικές κριτικές, 1 για θετικές κριτικές).

#### class LogisticRegression:

##### **Αρχικοποίηση (\_\_init\_\_ μέθοδος):**

Ο constructor αρχικοποιεί το μοντέλο λογιστικής παλινδρόμησης με διάφορες παραμέτρους όπως **learning\_rate**, **num\_iterations** (αριθμός επαναλήψεων εκπαίδευσης), **threshold** (decision boundary for classification) και **lambda\_param** (παραμέτρος κανονικοποίησης).

Ορίζουμε τις υπερπαραμέτρους σύμφωνα με την εκφώνηση :

- **m=1000**
- **n=0.9** (Λέξεις που εμφανίζονται σε περισσότερο από το 90% )(max\_df)

- `k=50 (min_df)`
- `lambda_param (λ) = 0.01` (L2 κανονικοποίηση)

### Συνάρτηση Σιγμοειδούς (sigmoid μέθοδος):

Η συνάρτηση **sigmoid** περιορίζει την έξοδο του μοντέλου στο διάστημα (0, 1), μετασχηματίζοντάς την σε πιθανότητες. Και ορίζεται ως  $\sigma(z) = \frac{1}{(1+e^{-z})}$ .

### Μέθοδος Προσαρμογής (fit μέθοδος):

Η μέθοδος **fit** εκπαιδεύει το μοντέλο λογιστικής παλινδρόμησης χρησιμοποιώντας την μέθοδο της κλίσης (Στοχαστική Ανάβαση Κλίσης). Παίρνει τα χαρακτηριστικά εισόδου `X_train_count` και `y_train`.

Αρχικοποιεί τα βάρη και την πόλωση σε μηδενικά. Κάνει επαναλήψεις για τον καθορισμένο αριθμό επαναλήψεων (`num_iterations`) για να ενημερώσει τα βάρη και την πόλωση βάσει της κλίσης της συνάρτησης απώλειας. Η συνάρτηση απώλειας περιλαμβάνει τη λογαριθμική απώλεια και τον όρο κανονικοποίησης **L2**.

### Μέθοδος Πρόβλεψης (predict μέθοδος):

Η μέθοδος **predict** παίρνει τα χαρακτηριστικά εισόδου (`X`) και υπολογίζει την έξοδο του μοντέλου χρησιμοποιώντας τα μαθημένα βάρη και το όριο απόφασης.

Η έξοδος περνά από τη συνάρτηση σιγμοειδούς για να πάρουμε πιθανότητες.

Οι πιθανότητες στη συνέχεια συγκρίνονται με ένα καθορισμένο όριο, και με βάση αυτήν τη σύγκριση, τα αποτελέσματα μετατρέπονται σε δυαδικές προβλέψεις, όπου οι τιμές πάνω από το όριο αντιστοιχούν στην ετικέτα 1, ενώ οι τιμές κάτω ή ίσες με το όριο αντιστοιχούν στην ετικέτα 0.

### Αξιολόγηση απόδοσης:

Υπολογίζεται και τυπώνεται ο πίνακας με το ποσοστό ορθότητας (`accuracy`) καθώς και τα `precision`, `recall`, `F1-score` για τα δεδομένα ελέγχου και εκπαίδευσης.

### Καμπύλες Μάθησης:

Οι λίστες (`train accuracies`, `test accuracies`, `precisions`, `recalls`, `f1_scores`) αρχικοποιούνται για την αποθήκευση τιμών.

Το `sizes` είναι μια λίστα που περιέχει διαφορετικά μεγέθη υποσυνόλων.

Ένας βρόχος (`for`) επαναλαμβάνεται σε διαφορετικά μεγέθη υποσυνόλου.

Για κάθε επανάληψη, δημιουργείται νέο instance του ταξινομητή `LogisticRegression` και εκπαιδεύεται σε ένα υποσύνολο των δεδομένων εκπαίδευσης (`X_train_binary[:size].toarray()` και `y_train[:size]`).

Στη συνέχεια γίνονται προβλέψεις τόσο στο υποσύνολο εκπαίδευσης όσο και στο υποσύνολο ελέγχου. Υπολογίζονται μετρήσεις accuracy, precision, recall, f1 και αποθηκεύονται στις αντίστοιχες λίστες.

Τα αντίστοιχα διαγράμματα εμφανίζονται χρησιμοποιώντας το plt.show().

### Μέρος Β' :

#### Naïve Bayes :

Στο αρχείο κώδικα **NaiveBayesSK.py** :

Χρησιμοποιούμε υλοποίηση του αλγορίθμου από την βιβλιοθήκη Sckit-learn (BernoulliNB())

Φαίνεται ότι η δική μας υλοποίηση του Naive Bayes και η υλοποίηση BernoulliNB του scikit-learn παρέχουν παρόμοια αποτελέσματα, αλλά υπάρχουν ορισμένες διαφορές .

NaiveBayes.py :

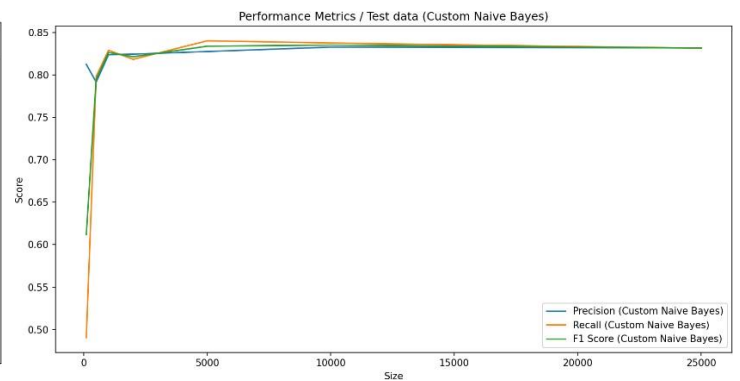
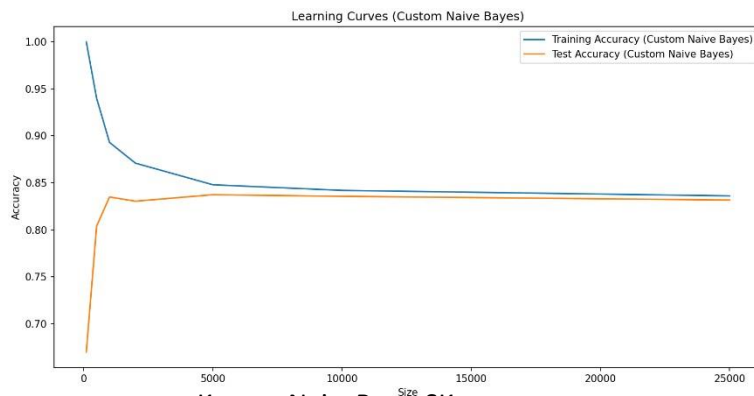
<b>Train Accuracy:</b> 0.83616	
<b>Test Accuracy:</b> 0.83176	
<b>Train Precision(0) :</b> 0.83	<b>Train Precision(1) :</b> 0.83
<b>Test Precision(0):</b> 0.83	<b>Test Precision(1) :</b> 0.83
<b>Train Recall (0):</b> 0.84	<b>Train Recall(1):</b> 0.84
<b>Test Recall(0) :</b> 0.83	<b>Test Recall(1):</b> 0.83
<b>Train F1-score(0) :</b> 0.84	<b>Train F1-score(1):</b> 0.84
<b>Test F1-score(0) :</b> 0.83	<b>Test F1-score(1):</b> 0.83

NaiveBayesSK.py:

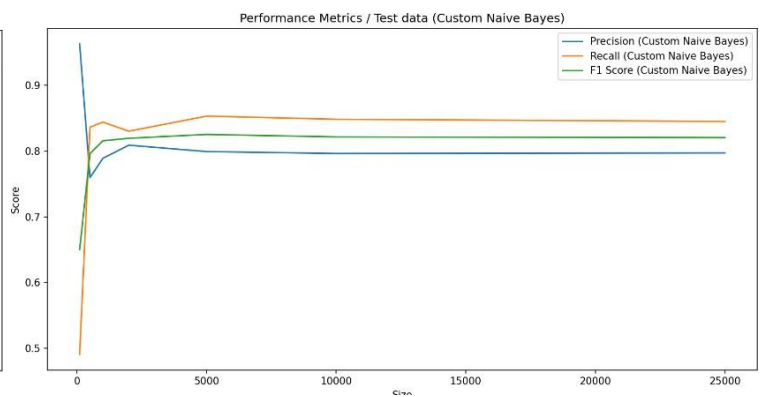
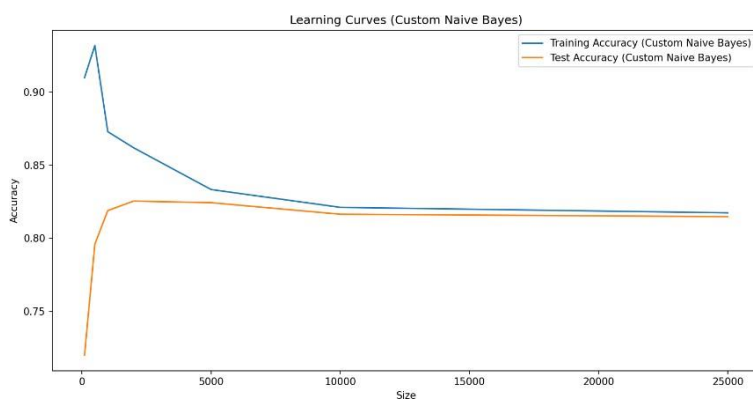
<b>Train Accuracy:</b> 0.81744	
<b>Test Accuracy:</b> 0.81484	
<b>Train Precision(0) :</b> 0.84	<b>Train Precision(1) :</b> 0.80
<b>Test Precision(0):</b> 0.83	<b>Test Precision(1) :</b> 0.80
<b>Train Recall (0):</b> 0.79	<b>Train Recall(1):</b> 0.85
<b>Test Recall(0) :</b> 0.78	<b>Test Recall(1):</b> 0.84
<b>Train F1-score(0) :</b> 0.81	<b>Train F1-score(1):</b> 0.82
<b>Test F1-score(0) :</b> 0.81	<b>Test F1-score(1):</b> 0.82


Η υλοποίηση του αλγορίθμου στο μέρος α' φαίνεται να έχει ελαφρώς υψηλότερη ακρίβεια τόσο σε σετ εκπαίδευσης όσο και σε σετ δοκιμών σε σύγκριση με την υλοποίηση του scikit-learn. Accuracy, precision, recall και F1 και στις δύο υλοποιήσεις είναι αρκετά κοντά, με την προσαρμοσμένη υλοποίηση να εμφανίζει ελαφρώς υψηλότερες τιμές.

Αντίστοιχα διαγράμματα για NaiveBayes.py :



Και για NaiveBayesSK.py :



 **RandomForest :**

Στο αρχείο κώδικα **RandomForestSK.py** :

Χρησιμοποιούμε υλοποίηση του αλγορίθμου από την βιβλιοθήκη Sckit-learn (RandomForestClassifier()) .

Φαίνεται ότι η υλοποίηση RandomForestClassifier του scikit-learn είναι πιο αποδοτική από τη δική μας υλοποίηση του Random Forest , κατά περίπου 7 μονάδες ως προς την ακρίβεια και σημαντική διαφορά ως προς την ταχύτητα εκτέλεσης.

RandomForest.py :

**Train Accuracy:** 0.67

**Test Accuracy:** 0.67

**Train Precision(0) :** 0.78

**Test Precision (0):** 0.79

**Train Recall(0) :** 0.48

**Test Recall(0) :** 0.48

**Train F1-score(0) :** 0.60

**Test F1-score(0) :** 0.60

**Train Precision(1) :** 0.63

**Test Precision(1) :** 0.63

**Train Recall(1) :** 0.86

**Test Recall (1) :** 0.87

**Train F1-score(1) :** 0.73

**Test F1-score(1) :** 0.73

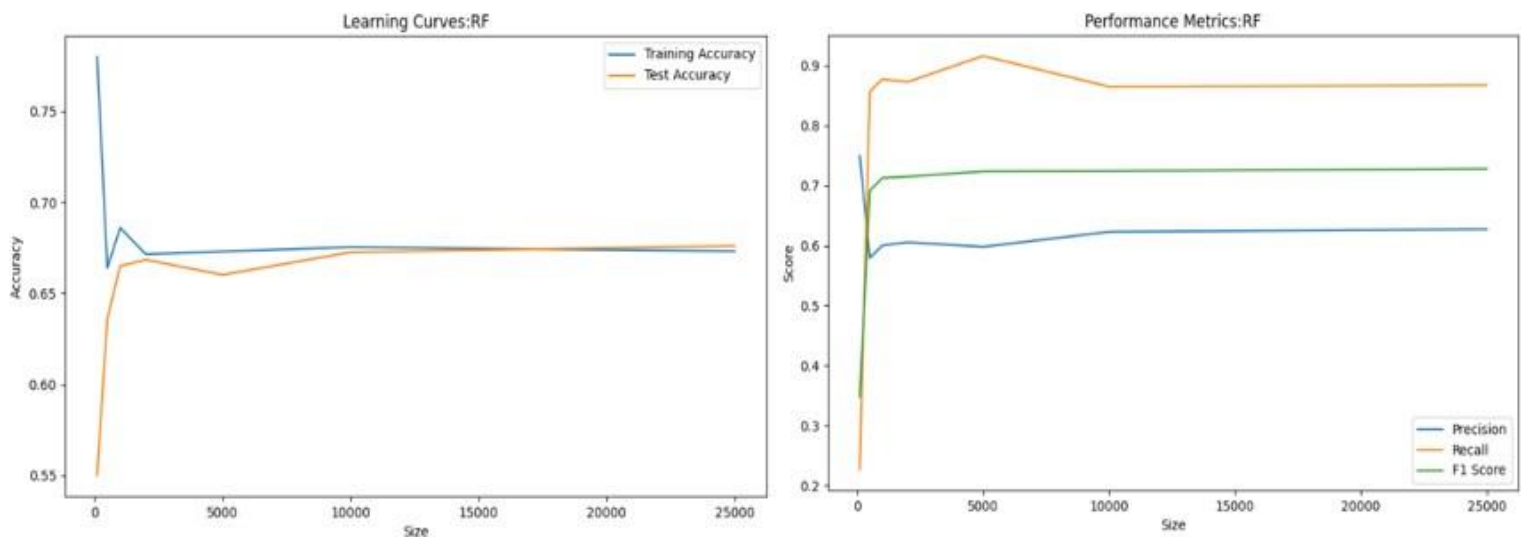
RandomForestSK.py:

**Train Accuracy:** 0.74  
**Test Accuracy:** 0.736

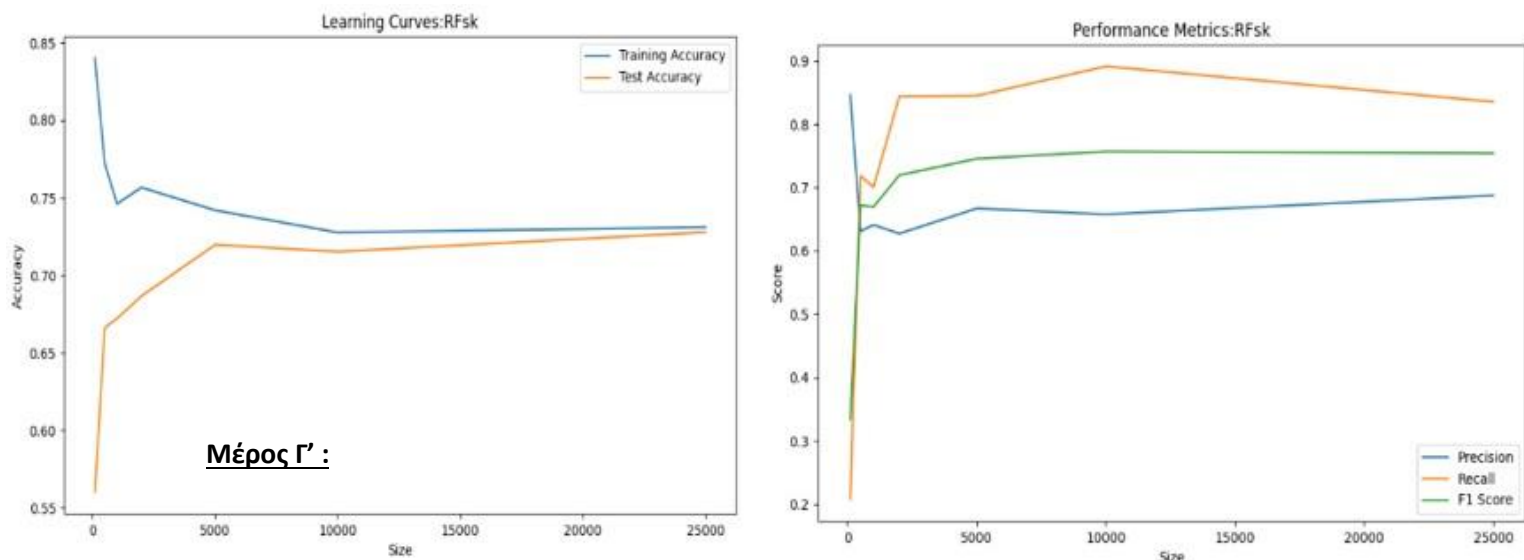
<b>Train Precision(0) :</b> 0.77	<b>Train Precision(1) :</b> 0.72
<b>Test Precision (0):</b> 0.77	<b>Test Precision(1) :</b> 0.71
<b>Train Recall(0) :</b> 0.69	<b>Train Recall(1) :</b> 0.79
<b>Test Recall(0) :</b> 0.68	<b>Test Recall (1) :</b> 0.79
<b>Train F1-score(0) :</b> 0.73	<b>Train F1-score(1) :</b> 0.75
<b>Test F1-score(0) :</b> 0.72	<b>Test F1-score(1) :</b> 0.75

Accuracy, precision, recall και F1 παρατηρείται ότι στην έτοιμη υλοποίηση είναι κυρίως μεταξύ 0.7 και 0.8 ενώ στην δική μας υλοποίηση υπάρχουν μεγαλύτερες ανωμαλίες(οι τιμές κυμαίνονται μεταξύ 0.48 και 0.87)

Αντίστοιχα διαγράμματα για RandomForest.py:



Και για RandomForestSK.py:





## Logistic Regression SKlearn:

Στο αρχείο κώδικα LogisticRegressionSK.py :

Χρησιμοποιούμε την έτοιμη υλοποίηση της λογιστικής παλινδρόμησης της βιβλιοθήκης Scikit-Learn. Τα αποτελέσματα της βιβλιοθήκης έχουν πολύ παρόμοια αποτελέσματα με την δική μας υλοποίηση, με πολύ παρόμοια απόδοση αλλά πολύ γρηγορότερη εκτέλεση του προγράμματος (running time) της sklearn. Από κάτω φαίνονται τα αποτελέσματα και των δυο προγραμμάτων:

### LogisticRegression.py :

**Train Accuracy:** 0.88720

**Test Accuracy:** 0.84980

**Train Precision(0) :** 0.90 **Train Precision(1) :** 0.88

**Test Precision (0):** 0.85 **Test Precision(1) :** 0.85

**Train Recall(0) :** 0.88 **Train Recall(1) :** 0.90

**Test Recall(0) :** 0.84 **Test Recall (1) :** 0.86

**Train F1-score(0) :** 0.89 **Train F1-score(1) :** 0.89

**Test F1-score(0) :** 0.84 **Test F1-score(1) :** 0.85

### LogisticRegressionSK.py

**Train Accuracy:** 0.88096

**Test Accuracy:** 0.87900

**Train Precision(0) :** 0.88 **Train Precision(1) :** 0.88

**Test Precision (0):** 0.88 **Test Precision(1) :** 0.88

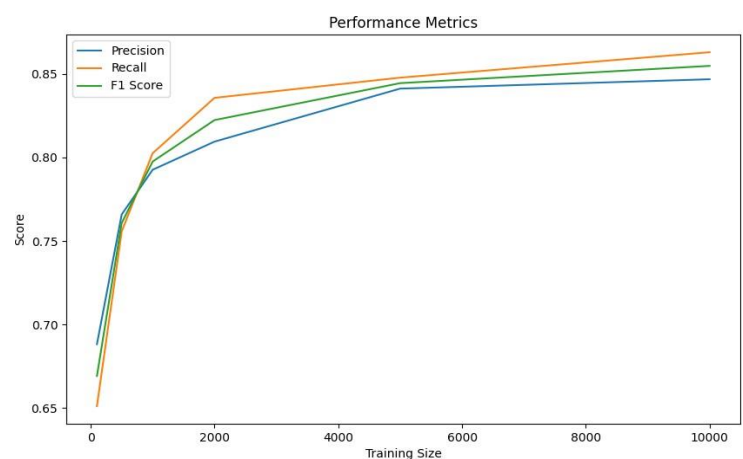
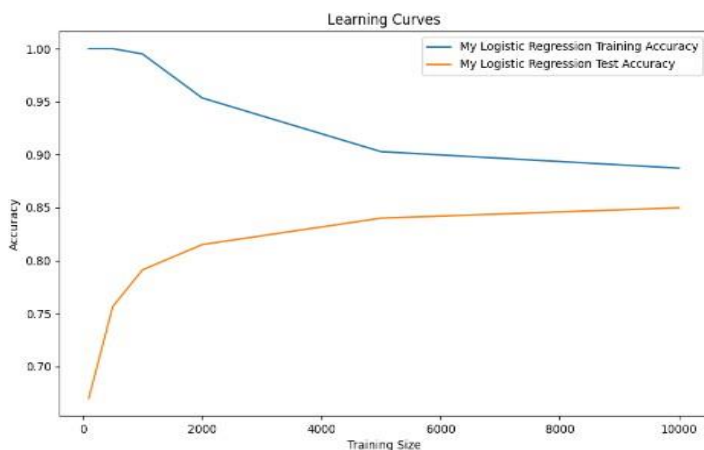
**Train Recall(0) :** 0.87 **Train Recall(1) :** 0.90

**Test Recall(0) :** 0.87 **Test Recall (1) :** 0.90

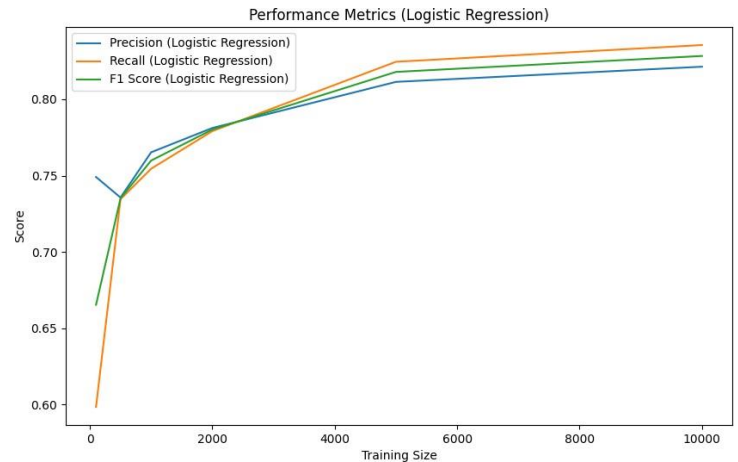
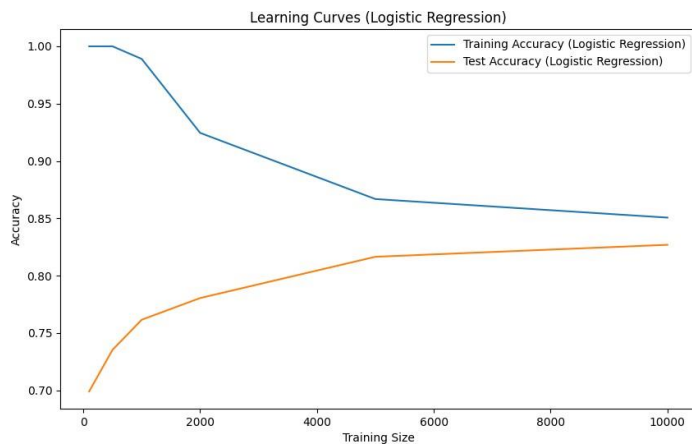
**Train F1-score(0) :** 0.87 **Train F1-score(1) :** 0.88

**Test F1-score(0) :** 0.87 **Test F1-score(1) :** 0.88

Διαγράμματα για myLogisticRegression.py :



Και για LogisticRegressionSKlearn.py :



### Μερος Γ:

Οι υπερπαραμέτροι που ορίζονται στον κώδικα είναι οι ακόλουθοι:

**num\_words:** Ο αριθμός των λέξεων που θέλουμε να κρατήσουμε από το σύνολο δεδομένων IMDB. Καθορίζει το λεξιλόγιο που θα χρησιμοποιηθεί στην ανάλυση.

**maxlen:** Το μέγιστο μήκος των ακολουθιών που θα προσαρμόσουμε. Αυτό είναι το μέγιστο μήκος της ακολουθίας που θα χρησιμοποιηθεί για την είσοδο των μοντέλων.

**output\_dim:** Το μέγεθος του διανύσματος κατάστασης ή τον αριθμό των νευρώνων σε κάθε επίπεδο ενσωματωμένης αναπαράστασης λέξεων. Αυτό καθορίζει τις διαστάσεις των ενσωματωμένων αναπαραστάσεων για κάθε λέξη.

**epochs:** Ο αριθμός των εποχών εκπαίδευσης, δηλαδή πόσες φορές θα δοθούν όλα τα δεδομένα εκπαίδευσης στο μοντέλο.

**batch\_size:** Το μέγεθος των παρτίδων δεδομένων που θα χρησιμοποιηθεί κάθε φορά κατά την εκπαίδευση. Κάθε παρτίδα αντιστοιχεί σε ένα σύνολο δεδομένων που προωθείται μαζί μέσω του δικτύου.

### **Multi-Layer Perceptron (MLP):**

MLP Train Accuracy: 0.8875200152397156

MLP Test Accuracy: 0.7770400047302246

	MLP	Naive Bayes	Naive Bayes SK	Random Forest	Random Forest SK	Logistic Regression	Logistic Regression SK
Train Accuracy	0.89	0.83	0.81	0.67	0.74	0.88	0.88
Test Accuracy	0.78	0.83	0.81	0.67	0.73	0.84	0.87

Το MLP έχει τα καλύτερα στατιστικά ακρίβειας ως προς το Train Accuracy.

Ωστόσο , ως προς το Test Accuracy ξεπερνάει μόνο το RandomForest

Το μοντέλο MLP πέτυχε πολύ υψηλή ακρίβεια (89% )στο σετ εκπαίδευσης αλλά δεν γενικεύτηκε καλά στο σύνολο επικύρωσης(78%), υποδεικνύοντας πιθανή υπερπροσαρμογή.

