



## Τεχνητή Νοημοσύνη (2022-2023)

Αλμπάνη Μάγδα (P3200005)

Βελισσαρίδης Γεώργιος (P3210255)

Ρηγάτος Διονύσιος (P3200262)

### Εργασία 2

## Εισαγωγή

Στην εργασία αυτή θα κάνουμε sentiment analysis πάνω σε κριτικές από το IMDb dataset του Keras με την χρήση αλγορίθμων μηχανικής μάθησης και νευρωνικών δικτύων που θα υλοποιήσουμε εμείς.

Επιλέξαμε εργαστούμε σε Python Jupyter Notebooks. Υλοποιήσαμε τους Bernoulli Naïve Bayes, ID3 και Logistic Regression και τους συγκρίνουμε με τις αντίστοιχες υλοποιήσεις της SKLearn (για το μέρος Β') των αλγορίθμων αυτών (BernoulliNB, DecisionTreeClassifier, SGDClassifier), καθώς και με ένα bi-directional GRU RNN που δημιουργήσαμε.

## Μέρος Α' & Β'

### 1. Προ-επεξεργασία

Για την προ-επεξεργασία ακολουθήσαμε τον κώδικα των φροντιστηρίων. Χρησιμοποιήσαμε τη συνάρτηση `tf.keras.datasets.imdb.load_data` για να αντλήσουμε τα reviews και να τα χωρίσουμε σε train και test δεδομένα, τα οποία όμως να περιέχουν μόνο τις τη συχνότερες λέξεις των δεδομένων εκπαίδευσης, πλην τις η πιο συχνές και η πιο σπάνιες από αυτές, που το πέτυχαμε περνώντας ως όρισμα για το `num_words` το `m-k` και για το `skip_top` το `n`. Οι τιμές των υπερπαραμέτρων αυτών είναι ελαφρώς διαφορετικές για κάθε αλγόριθμο και αναφέρονται παρακάτω. Τα reviews αποθηκεύονται στους πίνακες `x_train` και `x_test`, μεγέθους 25000 το καθένα, ενώ στα `y_train` και `y_test` αποθηκεύονται οι κατηγορίες του κάθε review (θετικό=1/αρνητικό=0). Στη συνέχεια,

μετατρέψαμε τα περιεχόμενα των `x_train` και `x_test` από λίστες αριθμών που ο καθένας αντιστοιχεί σε μία λέξη του λεξιλογίου, σε συμβολοσειρές λέξεων.

Τέλος, για να μπορούμε να αναλύσουμε τα δεδομένα μας με έναν δυαδικό ταξινομητή, τα περάσαμε από έναν binary CountVectorizer ώστε να μετατραπούν σε δυαδικά διανύσματα, όπου κάθε διάνυσμα αντιστοιχεί σε μια κριτική και κάθε τιμή αντιστοιχεί σε μία λέξη του λεξιλογίου μας που είναι 1 αν η συγκεκριμένη λέξη υπάρχει στο συγκεκριμένο review, και 0 διαφορετικά. Καταλήγουμε με τα εξής `x_train_binary` (`X`) και `y_train` (`y`):

$$X = \begin{bmatrix} \vec{x_1} \\ \vdots \\ \vec{x_t} \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_t \end{bmatrix}$$

Τέλος, ενσωματώσαμε information gain στους Naïve Bayes & Logistic Regression βάσει του φροντιστηρίου, με την επιλογή χρησιμοποίησής του να βρίσκεται μαζί με την δήλωση υπερ-παραμέτρων στην αρχή του notebook. Στις μετρήσεις μας το απενεργοποιήσαμε καθώς προτιμήσαμε μετρήσεις χωρίς το information gain.

## 2. Bernoulli Naïve Bayes

Θα υλοποιήσουμε την Bernoulli μορφή του αλγορίθμου Naïve Bayes με Laplace smoothing για την αποφυγή μηδενικών πιθανοτήτων. Επίσης θα χρησιμοποιήσουμε λογαριθμικές πιθανότητες έτσι ώστε να εξαλείψουμε το σενάριο σφαλμάτων λόγω υπερβολικά μικρών πολλαπλασιασμών.

Θα γίνει χρήση της βιβλιοθήκης `pumpry` και γραμμικής `algebra` για την αποφυγή από μία primitive υλοποίηση. Συγκεκριμένα, η βελτιωμένη υλοποίηση τρέχει σε λιγότερο από 2 δευτερόλεπτα έναντι στα 4 λεπτά μιας υλοποίησης με `for loops`.

Οι μέθοδοι της κλάσης και οι λειτουργίες τους είναι οι εξής:

```
def fit(self, X, y)
```

Η `fit` μας επιτρέπει να εκπαιδεύσουμε τον αλγόριθμό μας πάνω σε ένα σύνολο `train` δεδομένων (`X`) έχοντας τις ορθές αποκρίσεις τους (`y`).

Αρχικά υπολογίζει τις `a-priori` πιθανότητες για κάθε κλάση και λογαριθμίζει. Στη συνέχεια, υπολογίζει την πιθανότητα να είναι παρών ένα `feature` (δηλαδή το διάνυσμα χαρακτηριστικών να έχει τιμή 1 για αυτό το `feature`) δεδομένου ότι μια κριτική ανήκει σε μία κατηγορία. Αυτό το αποθηκεύουμε σε δύο μεταβλητές – την `r_log_pos_feature1` και την `r_log_neg_feature1` που περιέχουν την παραπάνω λογαριθμική πιθανότητα που υπολογίζεται ως εξής:

1. Αθροίζουμε κάθε στήλη του πίνακα με τα παραδείγματα εκπαίδευσης και καταλήγουμε με ένα διάνυσμα (μεγέθους ίσου με τον αριθμό των `features`) που περιέχει το πόσες φορές εμφανίστηκε κάθε `feature` σε μία συγκεκριμένη κατηγορία (π.χ. Θετική – 1 στο `r_log_pos_feature`).
2. Μετατρέπουμε κάθε τιμή του διανύσματος από άθροισμα σε πιθανότητα εμφάνισης του κάθε `feature` δεδομένης μίας κλάσης, διαιρώντας το αποτέλεσμα του βήματος (1) (+1) με το σύνολο των τιμών της κλάσης (+2) εφαρμόζοντας Laplace smoothing.

Έτσι καταλήγουμε με δύο διανύσματα, ένα για κάθε κλάση, όπου το καθένα περιέχει την πιθανότητα εμφάνισης ενός feature δεδομένου ότι ένα παράδειγμα με αυτό το feature ανήκει στην κλάση αυτή.

Ομοίως, υπολογίζουμε και δύο διανύσματα για κάθε κλάση, `r_log_pos_feature0` και `r_log_neg_feature0`, που το καθένα περιέχει την πιθανότητα μη εμφάνισης ενός feature (δηλαδή το διάνυσμα χαρακτηριστικών να έχει τιμή 0 για αυτό το feature), δεδομένου ότι ένα παράδειγμα χωρίς αυτό το feature ανήκει στην κλάση αυτή.

Αποθηκεύουμε τα αποτελέσματα αυτά και μπορούμε να τα χρησιμοποιήσουμε για την ταξινόμηση κριτικών.

```
def predict(self, X)
```

Η `predict` παίρνει ως είσοδο έναν πίνακα δυαδικών διανυσμάτων-κριτικών και μας επιτρέπει να χρησιμοποιήσουμε τα τέσσερα διανύσματα που υπολογίσαμε στην εκπαίδευση για να τα κατηγοριοποιήσουμε. Σε δύο μεταβλητές, τις `r_log_pos` και `r_log_neg`, αποθηκεύουμε διανύσματα που περιέχουν την λογαριθμική πιθανότητα κάθε κριτική να ανήκει σε μία κατηγορία δεδομένων των features της.

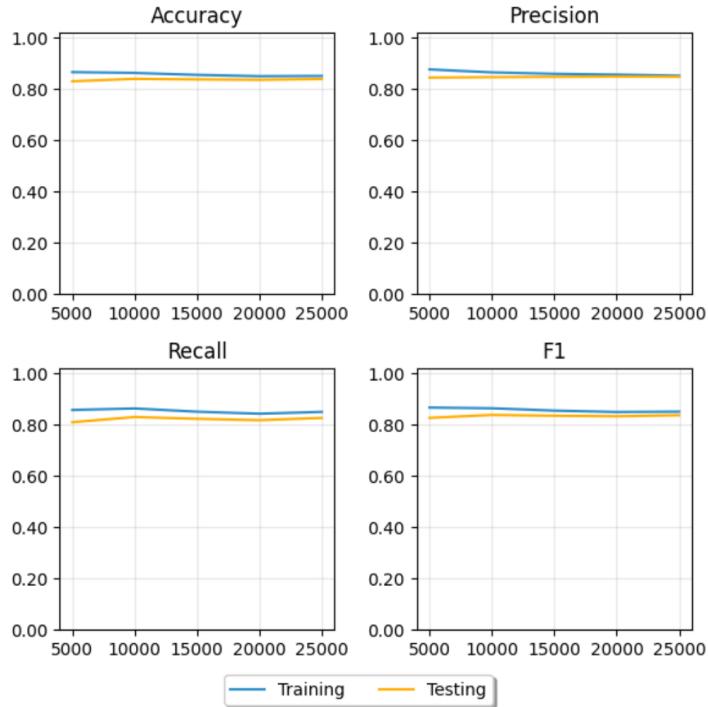
Το καταφέρνουμε υπολογίζοντας αρχικά έναν βοηθητικό πίνακα, τον `X_reverse`, τον οποίο δημιουργούμε αλλάζοντας κάθε στοιχείο του πίνακα με τα διανύσματα κριτικών (`X`) από 1 σε 0 και αντίστροφα. Στη συνέχεια αθροίζουμε μεταξύ τους δύο dot products: το dot product μεταξύ του `X_reverse`, ώστε να επεξεργαστούμε όλα τα features που είχαν αρχικά τιμή 0, και του πίνακα-διανύσματος που περιέχει την πιθανότητα κάθε feature να είναι 0, δεδομένου ότι η υπό εξέταση κριτική ανήκει σε μία κατηγορία (π.χ. `r_log_pos_feature0` για την θετική κατηγορία), και το dot product μεταξύ του `X`, ώστε να επεξεργαστούμε όλα τα features που έχουν τιμή 1, και του πίνακα-διανύσματος που περιέχει την πιθανότητα κάθε feature να είναι 1, δεδομένου ότι η υπό εξέταση κριτική ανήκει σε μία κατηγορία (π.χ. `r_log_pos_feature1` για την θετική κατηγορία). Στη συνέχεια προσθέτουμε την λογαριθμική `a-priori` πιθανότητα για κάθε κατηγορία στο παραπάνω άθροισμα, και καταλήγουμε με ένα διάνυσμα που περιέχει την πιθανότητα κάθε κριτική να ανήκει στην κατηγορία για την οποία κάνουμε τους υπολογισμούς. Αυτό γίνεται και για τις δύο κατηγορίες. ( $r = m-k-n$ ,  $r$  είναι λογαριθμικές πιθανότητες)

Αφού υπολογίσαμε την παραπάνω πιθανότητα, καταλήγουμε στην τελική `predicted` απόκριση του αλγορίθμου μας κατατάσσοντας κάθε κριτική στην κατηγορία 1 ή 0 (Θετική ή Αρνητική) ανάλογα ποια πιθανότητα είναι μεγαλύτερη στα δύο παραπάνω διανύσματα που υπολογίσαμε.

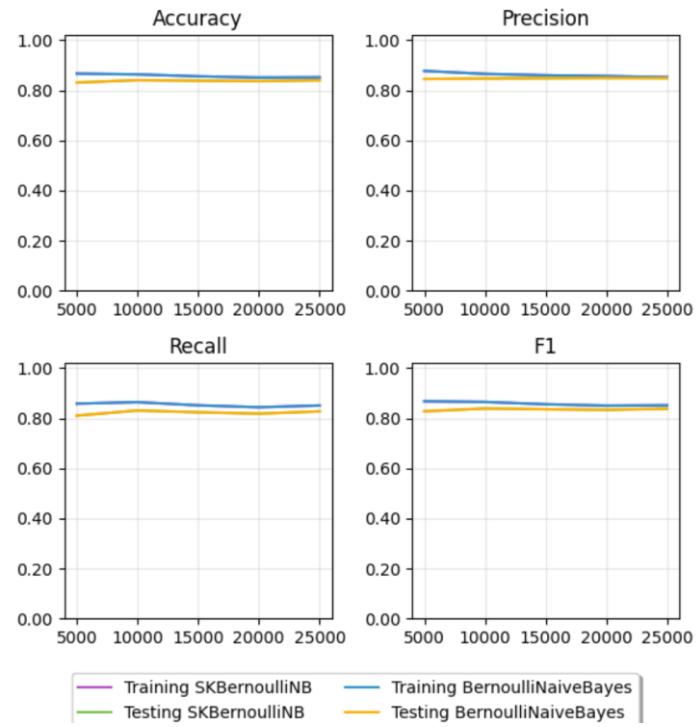
## Αποτελέσματα, Συγκρίσεις και Συμπεράσματα

Τα πειράματα που ακολουθούν έχουν τις εξής υπερπαραμέτρους: `m=2500`, `n=200`, `k=0`, `IG=off`. Ο αλγόριθμος τρέχει πανομοιότυπα, όσον αφορά τον χρόνο (1-3s) και τα αποτελέσματα, με τον `BernoulliNB` της `SKLearn` και εξίσου ικανοποιητικά σε σύγκριση με άλλους αλγορίθμους της. Αυτό είναι εμφανές, πέρα από τις καμπύλες, από το `table` διαφορών που δείχνει ακριβώς το πόσο καλύτερα η χειρότερα αποτελέσματα είχε ο αλγόριθμός μας, συγκριτικά με κάποιον άλλο (για έξτρα μετρήσεις βλέπε jupyter notebook).

## Learning Curve for BernoulliNaiveBayes



## Learning Curve Comparison for BernoulliNaiveBayes against SKBernoulliNB

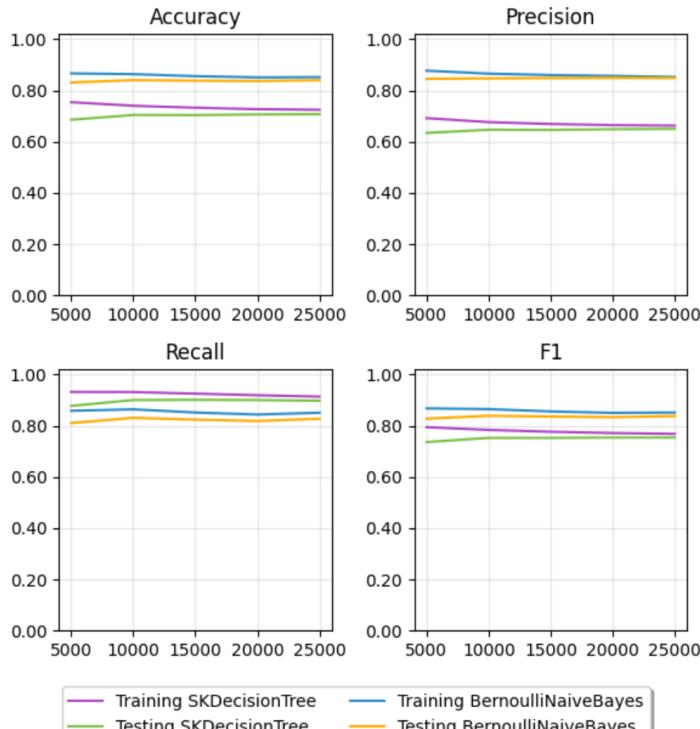


	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.87	0.83	0.88	0.85	0.86	0.81	0.87	0.83
10000	0.86	0.84	0.87	0.85	0.86	0.83	0.87	0.84
15000	0.86	0.84	0.86	0.85	0.85	0.82	0.86	0.84
20000	0.85	0.84	0.86	0.85	0.84	0.82	0.85	0.83
25000	0.85	0.84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

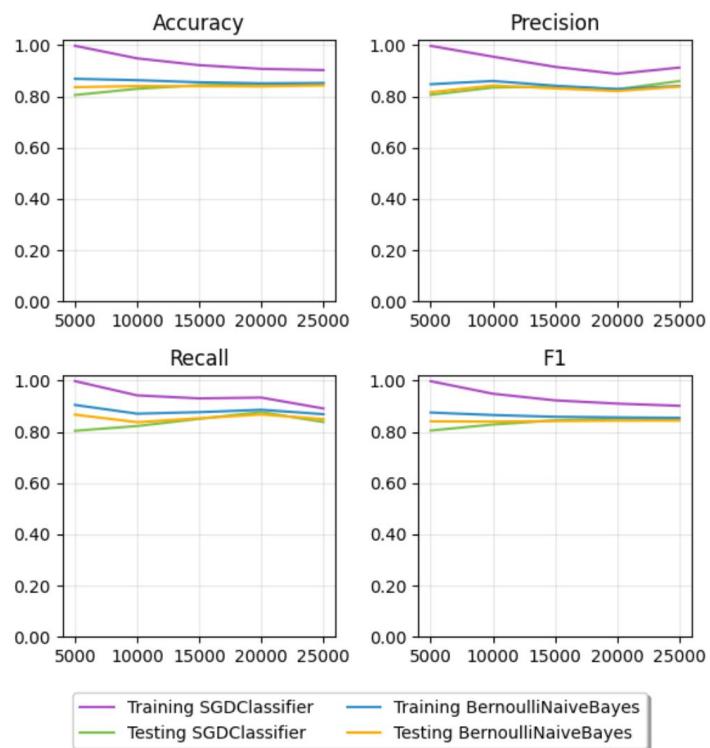
Classification Table Difference for BernoulliNaiveBayes against SKBernoulliNB								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
10000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
15000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
20000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	

We can see that there is virtually no difference with SKLearn's BernoulliNB

## Learning Curve Comparison for BernoulliNaiveBayes against SKDecisionTree

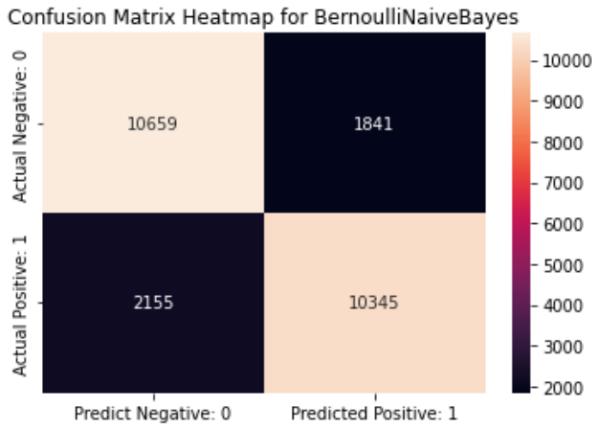


## Learning Curve Comparison for BernoulliNaiveBayes against SGDClassifier



	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.120000	0.140000	0.190000	0.220000	-0.070000	-0.070000	0.080000	0.090000
10000	0.120000	0.140000	0.190000	0.200000	-0.070000	-0.070000	0.090000	0.090000
15000	0.130000	0.140000	0.190000	0.200000	-0.080000	-0.080000	0.080000	0.090000
20000	0.120000	0.130000	0.200000	0.200000	-0.080000	-0.080000	0.080000	0.080000
25000	0.130000	0.130000	0.190000	0.200000	-0.060000	-0.070000	0.080000	0.090000

Classification Table Difference for BernoulliNaiveBayes against SGDClassifier								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	-0.130000	0.030000	-0.120000	0.040000	-0.130000	0.020000	-0.130000	0.030000
10000	-0.090000	0.010000	-0.090000	0.010000	-0.070000	0.020000	-0.070000	0.010000
15000	-0.060000	0.000000	-0.060000	0.010000	-0.070000	-0.030000	-0.060000	0.000000
20000	-0.060000	-0.010000	-0.060000	-0.010000	-0.060000	-0.020000	-0.060000	-0.020000
25000	-0.050000	-0.010000	-0.040000	0.010000	-0.070000	-0.040000	-0.050000	-0.010000



### 3. ID3

Θα υλοποιήσουμε τον αλγόριθμο ID3. Κατά την δημιουργία αντικειμένου τύπου ID3 πρέπει να δοθεί το μέγιστο βάθος του δένδρου που θέλουμε να δημιουργήσουμε. Για την δημιουργία των κόμβων του ID3 tree δημιουργήσαμε την κλάση Node(). Οι μέθοδοι της κλάσης και οι λειτουργίες τους είναι οι εξής:

```
def ID3_TREE(self, X, y, list_features, depth)
```

Ο ID3\_TREE δέχεται μια λίστα με παραδείγματα εκπαίδευσης, μια λίστα με την κατηγοριοποίηση του κάθε παραδείγματος (αρνητικό(0) ή θετικό(1)), μια λίστα με την κωδικοποίηση των features και το μέγιστο βάθος που θέλουμε να έχει το δένδρο μας. Επιστρέφει το δένδρο απόφασης.

- 1) Βρίσκει το information gain για κάθε feature και αποθηκεύει αυτό με το μεγαλύτερο. Επίσης, δημιουργεί έναν κόμβο root. Επιπλέον, δημιουργεί δυο νέες λίστες όπου η πρώτη περιέχει τα παραδείγματα αυτά που στην θέση του καλύτερου feature είχαν τιμή 0 ενώ τα άλλα αποθηκεύονται στην άλλη λίστα (τα παραδείγματα που αποθηκεύονται στις νέες λίστες δεν περιέχουν το καλύτερο feature). Οι λίστες αυτές δίνονται ως είσοδοι στην αναδρομική κλήση της συνάρτησης με σκοπό την δημιουργία του αριστερού και του δεξιού παιδιού του root.
- 2) Αν τα παραδείγματα ανήκουν κατά το 80% και άνω σε μια κατηγορία (θετικά ή αρνητικά) τότε δεν προχωράει σε αναδρομική κλήση αλλά δημιουργείται φύλλο στο δέντρο στο οποίο αποθηκεύεται η κατηγοριοποίηση των παραδειγμάτων. Με αυτό τον τρόπο επιτυγχάνουμε καλύτερη ικανότητα γενίκευσης και αποφεύγουμε το overfitting.
- 3) Αν το βάθος του δέντρου που δημιουργείται φτάσει στο μέγιστο βάθος που ζητείται τότε δεν προχωράει σε αναδρομική κλήση αλλά δημιουργεί φύλλο στο δέντρο και κατηγοριοποιεί τα παραδείγματα με βάση την πλειοψηφία. Στην περίπτωση όπου τα μισά παραδείγματα κατηγοριοποιούνται ως θετικά και τα αλλά μισά ως αρνητικά τότε η απόφαση για αυτά τα παραδείγματα λαμβάνεται με τυχαίο τρόπο.

```
def fit(self, X, y)
```

Η fit δέχεται μια λίστα με παραδείγματα εκπαίδευσης και μια λίστα με την κατηγοριοποίηση του κάθε παραδείγματος. Καλεί την συνάρτηση ID3\_TREE και αποθηκεύει το δένδρο που επιστρέφει σε μια μεταβλητή root της κλάσης ID3.

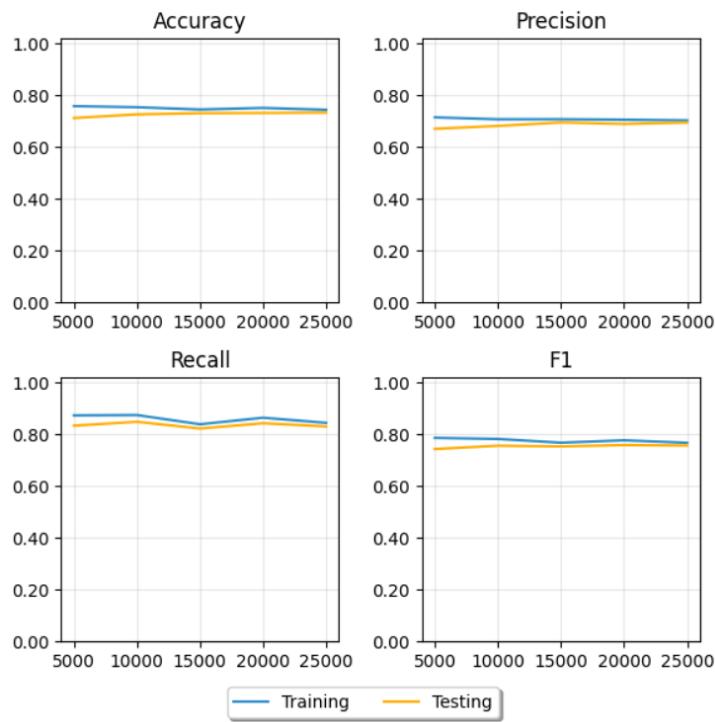
```
def predict(self, X)
```

H predict παίρνει ως είσοδο έναν πίνακα δυαδικών διανυσμάτων-κριτικών και μας επιτρέπει να χρησιμοποιήσουμε το δένδρο που δημιουργήσαμε στην εκπαίδευση για να τα κατηγοριοποιήσουμε. Στην ρίζα του δέντρου βρίσκεται το feature με το μεγαλύτερο information gain οπότε για κάθε διάνυσμα ελέγχει αν στην θέση αυτού του feature έχει τιμή 0 ή 1. Αν έχει μηδέν συνεχίζει να ψάχνει στο αριστερό υποδένδρο αλλιώς συνεχίζει να ψάχνει στο δεξί υποδένδρο. Η διαδικασία επαναλαμβάνεται μέχρι να φτάσει σε φύλο όπου τελικά θα καταφέρει να κατηγοριοποιήσει το διάνυσμα(κριτική) αυτό με την τιμή που είναι αποθηκευμένη σε αυτό το φύλο 1 ή 0 (Θετική ή Αρνητική).

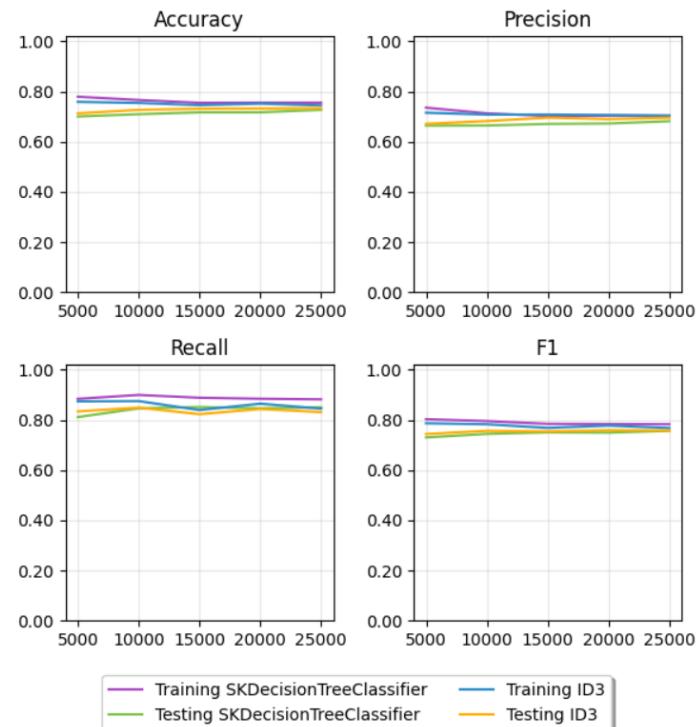
## Αποτελέσματα, Συγκρίσεις και Συμπεράσματα

Δίνοντας τις εξής τιμές στις υπερπαραμέτρους: m=2000, n=65,k=0 και depth =10 Τα αποτελέσματα του αλγορίθμου είναι σε μεγάλο βαθμό κοντά στα αποτελέσματα του DecisionTreeClassifier(criterion='entropy').

Learning Curve for ID3



Learning Curve Comparison for ID3 against SKDecisionTreeClassifier

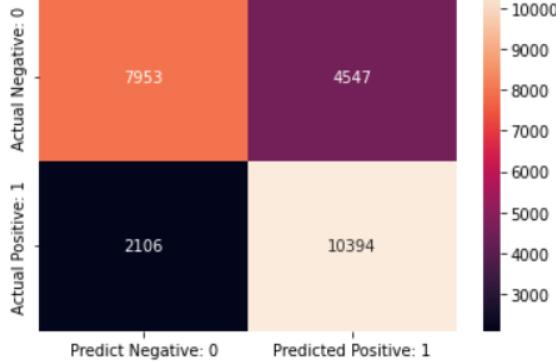


Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	0.76	0.71	0.72	0.67	0.87	0.83	0.79	0.74
10000	0.75	0.73	0.71	0.68	0.88	0.85	0.78	0.76
15000	0.75	0.73	0.71	0.70	0.84	0.82	0.77	0.75
20000	0.75	0.73	0.71	0.69	0.87	0.84	0.78	0.76
25000	0.74	0.73	0.70	0.70	0.85	0.83	0.77	0.76

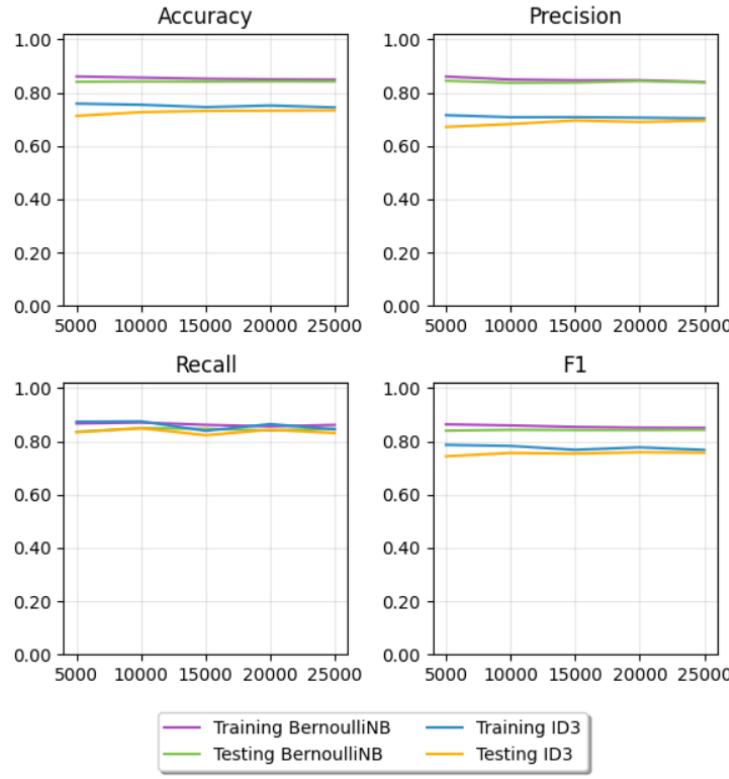
Classification Table Difference for ID3 against SKDecisionTreeClassifier								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	-0.020000	0.010000	-0.020000	0.010000	-0.010000	0.020000	-0.010000	0.010000
10000	-0.020000	0.020000	0.000000	0.010000	-0.020000	0.000000	-0.020000	0.010000
15000	-0.010000	0.010000	0.010000	0.030000	-0.050000	-0.030000	-0.010000	0.000000
20000	-0.010000	0.010000	0.010000	0.020000	-0.020000	-0.010000	0.000000	0.010000
25000	-0.020000	0.000000	0.000000	0.020000	-0.030000	-0.020000	-0.010000	0.000000

We can see that there is virtually no difference with SKLearn's DecisionTreeClassifier

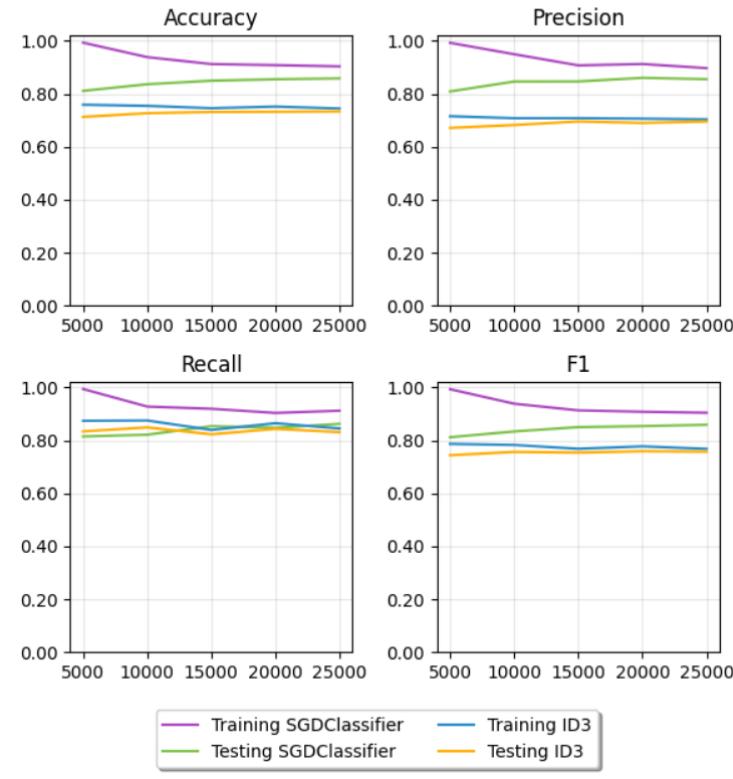
Confusion Matrix Heatmap for ID3



## Learning Curve Comparison for ID3 against BernoulliNB



## Learning Curve Comparison for ID3 against SGDClassifier



## 4. Logistic Regression

Η υλοποίηση της λογιστικής παλινδρόμησης απαιτησε τη δημιουργία εκτός του συνόλου εκπαίδευσης και ελέγχου, και ενός συνόλου επαλήθευσης για την εύρεση της τιμής της σταθεράς κανονικοποίησης  $\lambda$ .

Για να το επιτύχουμε αυτό, μετά την άντληση των δεδομένων από τη μέθοδο `load_data`, αποκόψαμε ένα μικρό κομμάτι από τα δεδομένα εκπαίδευσης ( $10\% = 2500$  reviews) με τη μέθοδο `train_test_split` του SKLearn. Στη συνέχεια, τα περάσαμε κι αυτά από τον `CountVectorizer` και τα χρησιμοποιήσαμε ως δεδομένα επαλήθευσης.

Η υλοποίηση του ίδιου του αλγορίθμου περιλαμβάνει τις μεθόδους `fit` και `predict`, όπως και οι προηγούμενοι αλγόριθμοι, καθώς και τη μέθοδο `prob_pos`.

Η `prob_pos` δέχεται ως όρισμα το διάνυσμα ιδιοτήτων ενός παραδείγματος και υπολογίζει τη λογιστική/σιγμοειδή συνάρτηση για το εσωτερικό γινόμενο του διανύσματος ιδιοτήτων και του διανύσματος βαρών, και το αποτέλεσμα αναπαριστά την πιθανότητα το συγκεκριμένο παράδειγμα να ανήκει στη θετική κατηγορία.

Η `fit` αρχικά προσθέτει στην αρχή κάθε διανύσματος ιδιοτήτων μία επιπλέον ιδιότητα ίση με 1, που αντιστοιχεί στο bias term και χρειάζεται για τη σωστή ενημέρωση των βαρών. Τα βάρη (και το bias term, που βρίσκεται στην πρώτη θέση του διανύσματος των βαρών) αρχικοποιούνται με 0. Στη συνέχεια αρχίζουμε να εκτελούμε στοχαστική κατάβαση κλίσης (ισοδύναμο με την ανάβαση, βλ. παρακάτω), κάνοντας `shuffle` τα παραδείγματα σε κάθε εποχή. Ενημερώνουμε τα βάρη με βάση τον τύπο:

$$\vec{w} \leftarrow (1 - 2 \cdot \lambda \cdot \eta) \cdot \vec{w} + \eta \cdot [y^{(i)} - P(c_+ | \vec{x}^{(i)})] \vec{x}^{(i)}$$

όπου  $w$  = το διάνυσμα βαρών,  $\lambda$  = η σταθερά κανονικοποίησης,  $\eta$  = το learning rate,  $y^{(i)}$  = η σωστή απόκριση για το παράδειγμα  $i$ ,  $P(c_+ | x^{(i)})$  = η πιθανότητα το παράδειγμα  $i$  να ανήκει στη θετική κατηγορία και  $x^{(i)}$  = το διάνυσμα χαρακτηριστικών του παραδείγματος  $i$ .

Χρησιμοποιούμε πολλαπλασιασμό πινάκων που επιταχύνει δραματικά την ταχύτητα της εκπαίδευσης. Η τιμή του learning rate η ισούται με 0.01, και καταλήξαμε σε αυτή μετά από πειράματα και έρευνα, όπως και για τις άλλες υπερ-παραμέτρους του αλγορίθμου.

Στη θεωρία είδαμε ότι ο στόχος στη λογιστική παλινδρόμηση είναι η μεγιστοποίηση της πιθανοφάνειας, αλλά αυτό είναι ισοδύναμο με το την ελαχιστοποίηση του αντίθετου της πιθανοφάνειας δια τον αριθμό των παραδειγμάτων, μέγεθος που είναι γνωστό ως log loss και έχει την παρακάτω μορφή:

$$Logloss = -\frac{1}{N} \sum_{i=1}^N [y_i \ln p_i + (1 - y_i) \ln(1 - p_i)]$$

όπου  $N$  = ο αριθμός παραδειγμάτων,  $y_i$  = η σωστή απόκριση για το παράδειγμα  $i$  και  $p_i$  = η πιθανότητα το παράδειγμα  $i$  να ανήκει στη θετική κατηγορία.

Εφόσον λοιπόν προσπαθούμε να ελαχιστοποιήσουμε το log loss, δεν εκτελούμε ανάβαση, αλλά κατάβαση κλίσης.

Στη υλοποίησή μας προσθέσαμε early stopping, που απαίτησε τον υπολογισμό του log loss σε κάθε εποχή και την αποθήκευση της χαμηλότερης τιμής που έχει λάβει σε προηγούμενη εποχή. Η συνθήκη τερματισμού είναι το log loss να είναι μεγαλύτερο από την ελάχιστη τιμή του συν έναν όρο tolerance (=1e-3) για πέντε φορές πριν η ελάχιστη τιμή του μειωθεί σε κάποια εποχή (στην οποία περίπτωση, ο μετρητής μηδενίζεται). Αν η συνθήκη τερματισμού του early stopping δεν ικανοποιηθεί εγκαίρως, η στοχαστική κατάβαση κλίσης θα σταματήσει μετά από max\_epochs αριθμό εποχών (=1000).

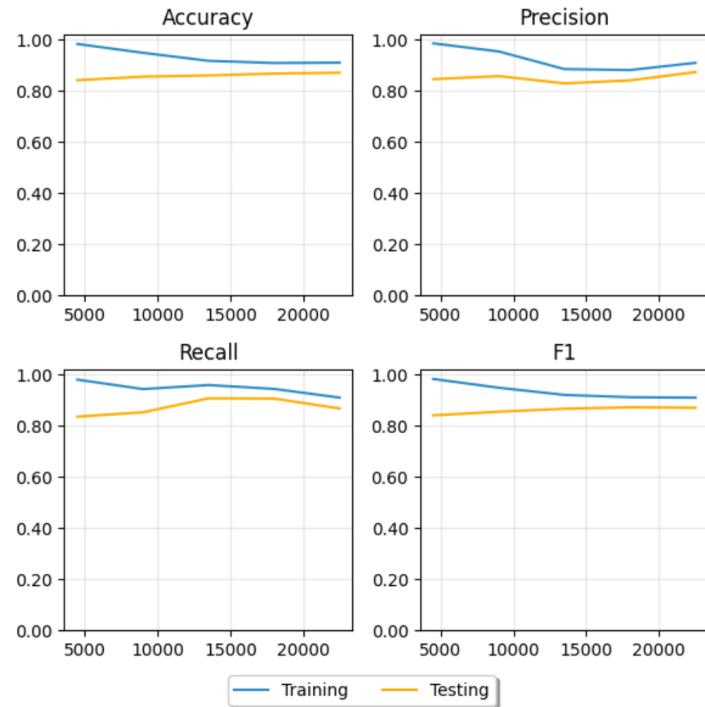
Η συνάρτηση predict δέχεται ως όρισμα ένα διάνυσμα παραδειγμάτων, σε καθένα απ' τα οποία προσθέτει μία ιδιότητα ίση με 1 στην αρχή τους, που αντιστοιχεί στο bias term. Στη συνέχεια, για κάθε διάνυσμα παραδειγμάτων υπολογίζει το εσωτερικό γινόμενό του με το διάνυσμα βαρών που προέκυψε από την εκπαίδευση του αλγορίθμου και αν αυτό είναι θετικό τότε προσθέτει 1 στο διάνυσμα των αποκρίσεων, διαφορετικά προσθέτει 0. Αφού κατατάξουμε όλα τα παραδείγματα, επιστρέφουμε το διάνυσμα των αποκρίσεων.

Τέλος, επιλέξαμε πειραματικά την καλύτερη τιμή για τη σταθερά κανονικοποίησης  $\lambda$ , μεταξύ διαθέσιμων τιμών που συνήθως λαμβάνει, 0.01, 0.005, 0.001, 0.0005 και 0.0001. Αναλυτικότερα, κάναμε fit τον αλγόριθμο σε όλα τα δεδομένα εκπαίδευσης για κάθε τιμή  $\lambda$ , και ελέγχαμε το accuracy score στα δεδομένα επικύρωσης. Κρατήσαμε και χρησιμοποιήσαμε στα υπόλοιπα μέρη, την τιμή  $\lambda$  που είχε το καλύτερο accuracy\_score (παρατηρήσαμε ότι προκύπτει το 0.001 ή 0.005, με τη μη σταθερότητα του αποτελέσματος να οφείλεται στη στοχαστικότητα του αλγορίθμου).

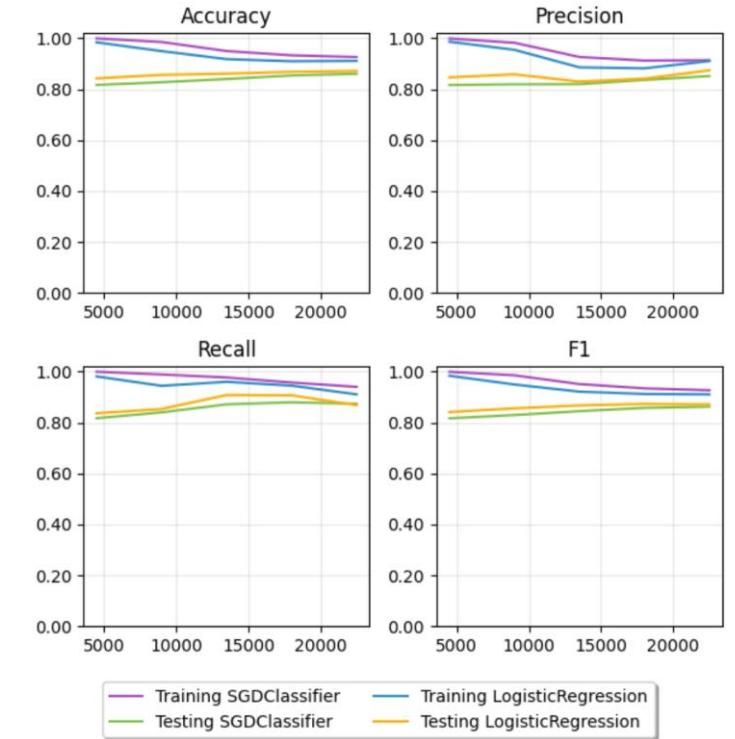
## Αποτελέσματα, Συγκρίσεις και Συμπεράσματα

Τα αποτελέσματα που ακολουθούν χρησιμοποιούν τις εξής τιμές υπερ-παραμέτρων:  $m=3000$ ,  $n=20$ ,  $k=20$ ,  $IG=off$ . Ο αλγόριθμος τρέχει περίπου 3 φορές πιο αργά από τον SGDClassifier της SKLearn, αλλά επιτυγχάνει καλύτερη επίδοση σε δεδομένα ελέγχου, καθώς και λιγότερο overfitting. Παρουσιάζει επίσης καλύτερη επίδοση, τόσο σε δεδομένα εκπαίδευσης όσο και ελέγχου σε σχέση με τους άλλους αλγορίθμους με τους οποίους συγκρίνουμε, που γίνεται φανερό από τις καμπύλες αλλά και τους πίνακες σύγκρισης.

Learning Curve for LogisticRegression



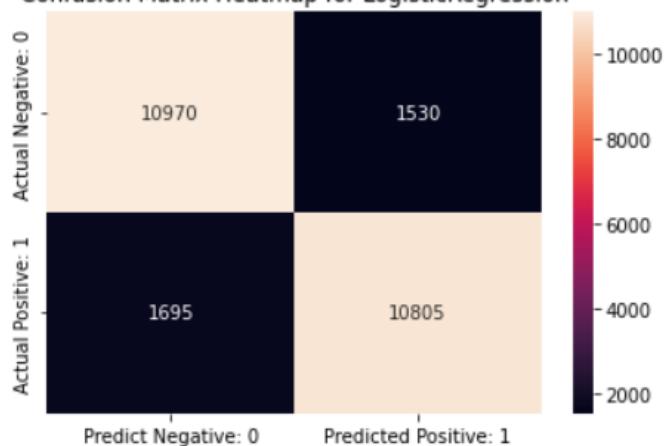
Learning Curve Comparison for LogisticRegression against SGDClassifier



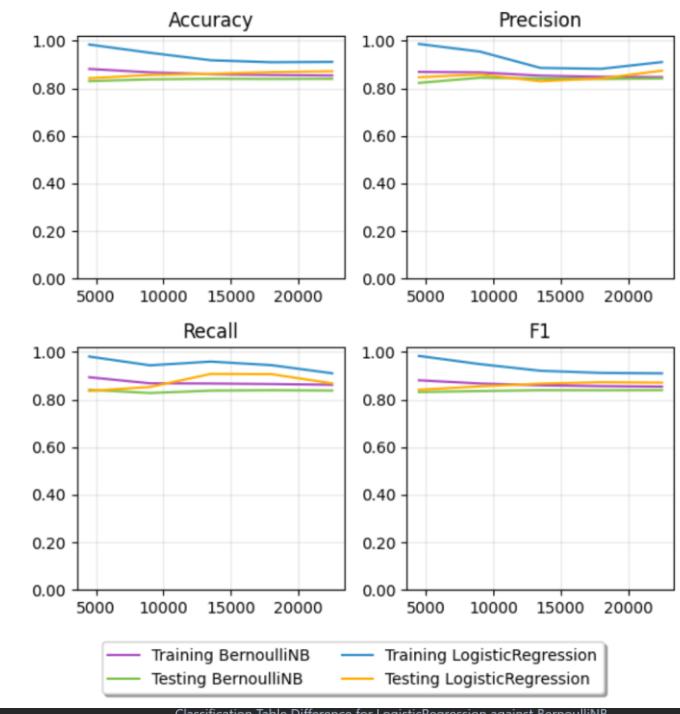
	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
4500	0.98	0.84	0.99	0.85	0.98	0.84	0.98	0.84
9000	0.95	0.86	0.96	0.86	0.94	0.85	0.95	0.86
13500	0.92	0.86	0.89	0.83	0.96	0.91	0.92	0.87
18000	0.91	0.87	0.88	0.84	0.95	0.91	0.91	0.87
22500	0.91	0.87	0.91	0.87	0.91	0.87	0.91	0.87

Classification Table Difference for LogisticRegression against SGDClassifier								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
4500	-0.020000	0.020000	-0.010000	0.030000	-0.020000	0.020000	-0.020000	0.030000
9000	-0.040000	0.030000	-0.020000	0.040000	-0.050000	0.010000	-0.040000	0.030000
13500	-0.030000	0.020000	-0.040000	0.010000	-0.020000	0.040000	-0.030000	0.020000
18000	-0.020000	0.020000	-0.030000	0.000000	-0.010000	0.030000	-0.020000	0.010000
22500	-0.020000	0.010000	0.000000	0.020000	-0.030000	0.000000	-0.020000	0.010000

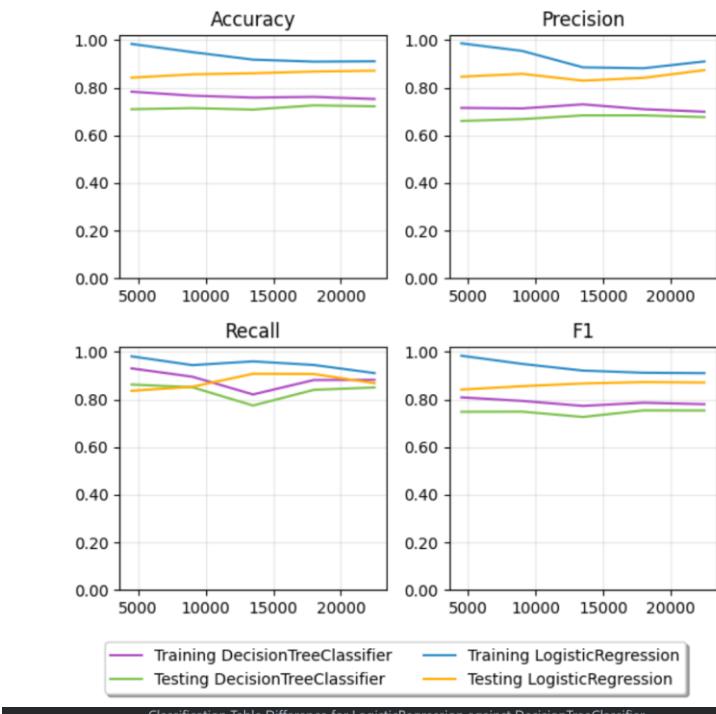
Confusion Matrix Heatmap for LogisticRegression



Learning Curve Comparison for LogisticRegression against BernoulliNB



Learning Curve Comparison for LogisticRegression against DecisionTreeClassifier



Classification Table Difference for LogisticRegression against BernoulliNB								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
4500	0.100000	0.010000	0.120000	0.030000	0.090000	0.000000	0.100000	0.010000
9000	0.080000	0.020000	0.090000	0.020000	0.070000	0.020000	0.080000	0.020000
13500	0.060000	0.020000	0.040000	-0.010000	0.090000	0.070000	0.060000	0.030000
18000	0.050000	0.030000	0.030000	0.000000	0.080000	0.070000	0.050000	0.030000
22500	0.060000	0.030000	0.060000	0.030000	0.050000	0.030000	0.060000	0.030000

Classification Table Difference for LogisticRegression against DecisionTreeClassifier								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
4500	0.200000	0.130000	0.270000	0.190000	0.050000	-0.020000	0.170000	0.090000
9000	0.180000	0.150000	0.250000	0.190000	0.040000	0.000000	0.160000	0.110000
13500	0.160000	0.150000	0.160000	0.150000	0.140000	0.130000	0.150000	0.140000
18000	0.150000	0.140000	0.170000	0.160000	0.070000	0.070000	0.120000	0.120000
22500	0.160000	0.150000	0.210000	0.190000	0.030000	0.020000	0.130000	0.120000

## Μέρος Γ'

Για το μέρος αυτό αποφασίσαμε να φτιάξουμε ένα bi-directional GRU RNN, βασισμένοι στον αντίστοιχο κώδικα του εργαστηρίου.

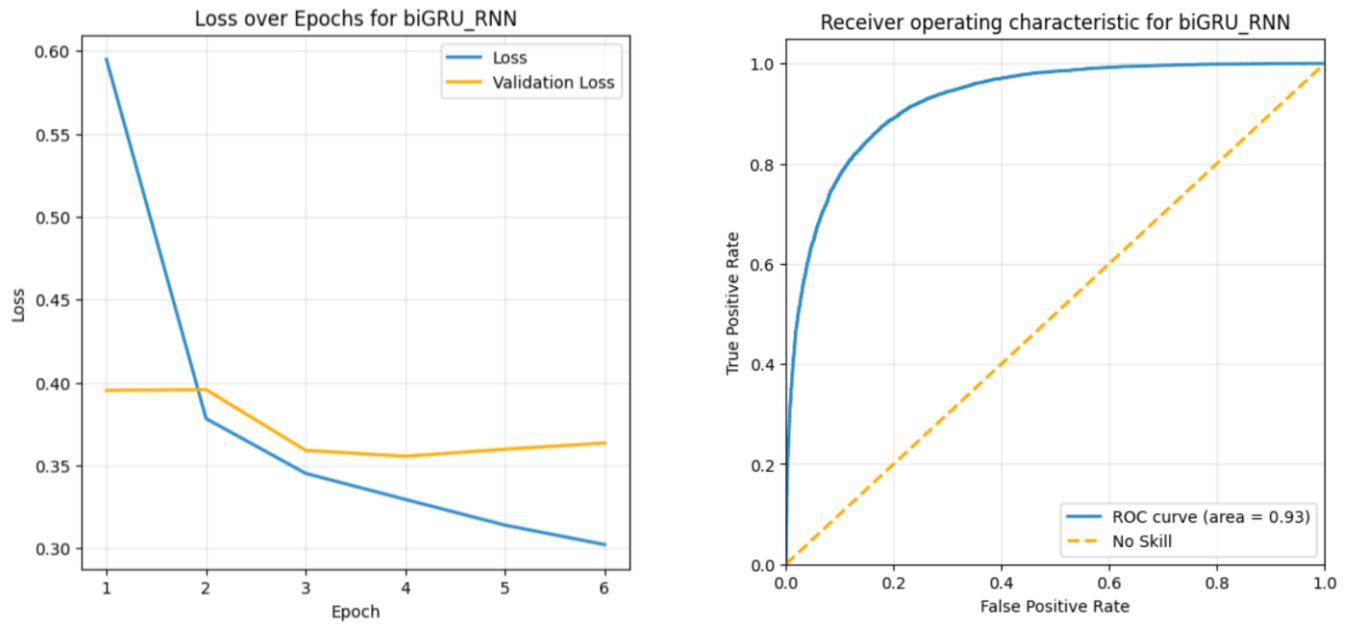
Αφού αντλήσαμε τα δεδομένα εκπαίδευσης και ελέγχου από τη μέθοδο load\_data, μετατρέψαμε τα περιεχόμενα των x\_train και x\_test σε συμβολοσειρές με τρόπο πανομοιότυπο με αυτόν του μέρους Α'. Στη συνέχεια, χρησιμοποιήσαμε τον κώδικα του εργαστηρίου για τη δημιουργία ενός TextVectorization layer, προκειμένου να απεικονίσουμε κάθε λέξη του λεξιλογίου μας σε έναν ακέραιο, για τελική χρήση στις ενθέσεις λέξεων.

Η δομή του RNN ξεκινά με ένα input layer, που δέχεται ένα review σε μορφή συμβολοσειράς. Ακολουθεί το TextVectorization layer που δημιουργήσαμε παραπάνω, το οποίο μετατρέπει τη συμβολοσειρά σε διάνυσμα ακεραίων, καθένας απ' τους οποίους αντιστοιχεί σε μία λέξη του review. Η διάσταση του διανύσματος είναι 240, τιμή που επιλέχτηκε ευρετικά ως ο μέσος αριθμός λέξεων των δεδομένων εκπαίδευσης, όπως και στο εργαστήριο. Το διάνυσμα αυτό περνά από το layer που δημιουργεί τις ενθέσεις των λέξεων εισόδου, ενώ καθεμία από τις ενθέσεις έχει μέγεθος 64. Τα επόμενα βήματα είναι να περάσουμε τα embeddings από λ layers από bi-directional RNNs, τις 128 εξόδους τους από ένα Dropout layer και τέλος από ένα Dense layer με μία έξοδο και συνάρτηση ενεργοποίησης τη σιγμοειδή, που μας δίνει την πιθανότητα το συγκεκριμένο review να είναι θετικό.

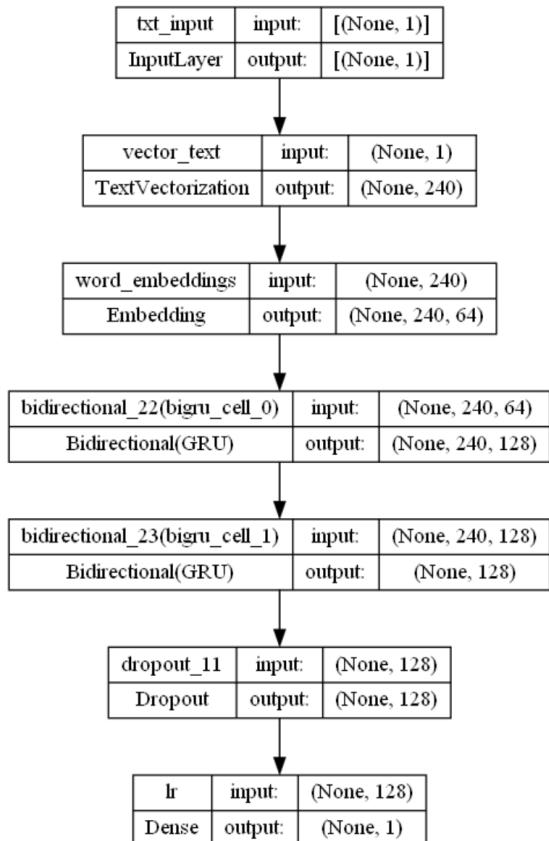
Οι πίνακες και οι καμπύλες χαρακτηριστικών δημιουργήθηκαν με τρόπο αντίστοιχο με τα προηγούμενα μέρη. Προστέθηκαν και καμπύλες για το loss στα δεδομένων εκπαίδευσης και επαλήθευσης καθώς και Receiver Operating Characteristic plot.

## Αποτελέσματα, Συγκρίσεις και Συμπεράσματα

Μετά από δοκιμές, καταλήξαμε στις εξής τιμές για τις υπερ-παραμέτρους σχετικές με τον αριθμό λέξεων/ιδιοτήτων που αντλούμε από το IMDB Dataset:  $m = 1000$ ,  $n = 20$ ,  $k = 0$ . Σε όλες τις μετρήσεις έχουμε  $\text{batch\_size}=64$ ,  $\text{emb\_size}=64$ ,  $\text{h\_size}=64$ , 6 epochs και 1 layer.

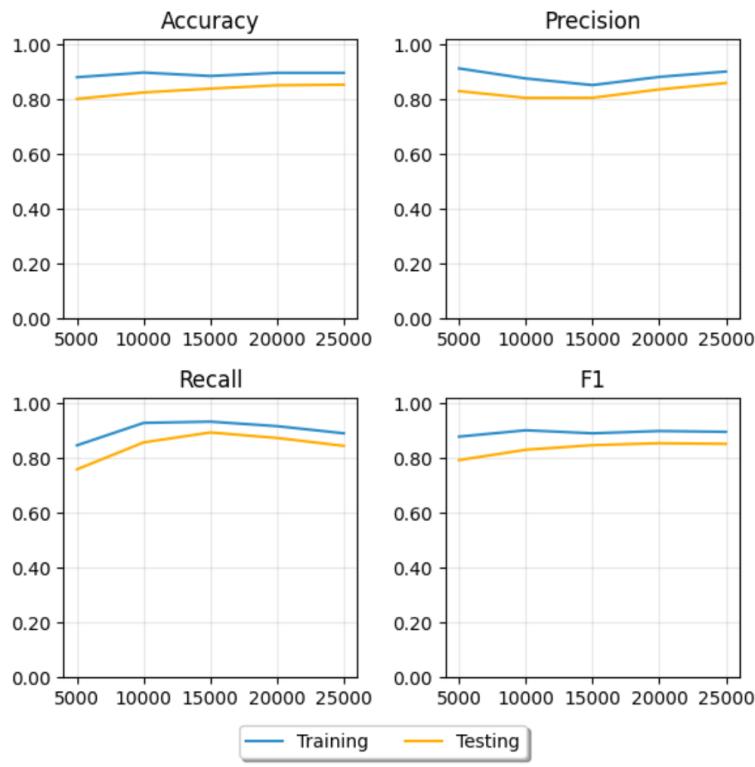


For all data – validation data is 20% of training data

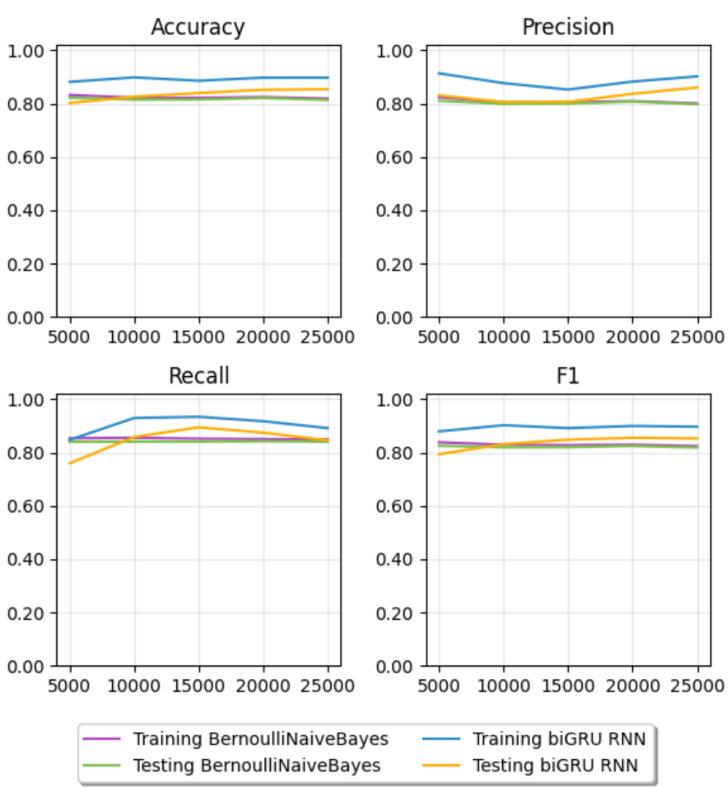


BiGRU RNN with 2 bidirectional GRU layers

## Learning Curve for biGRU RNN



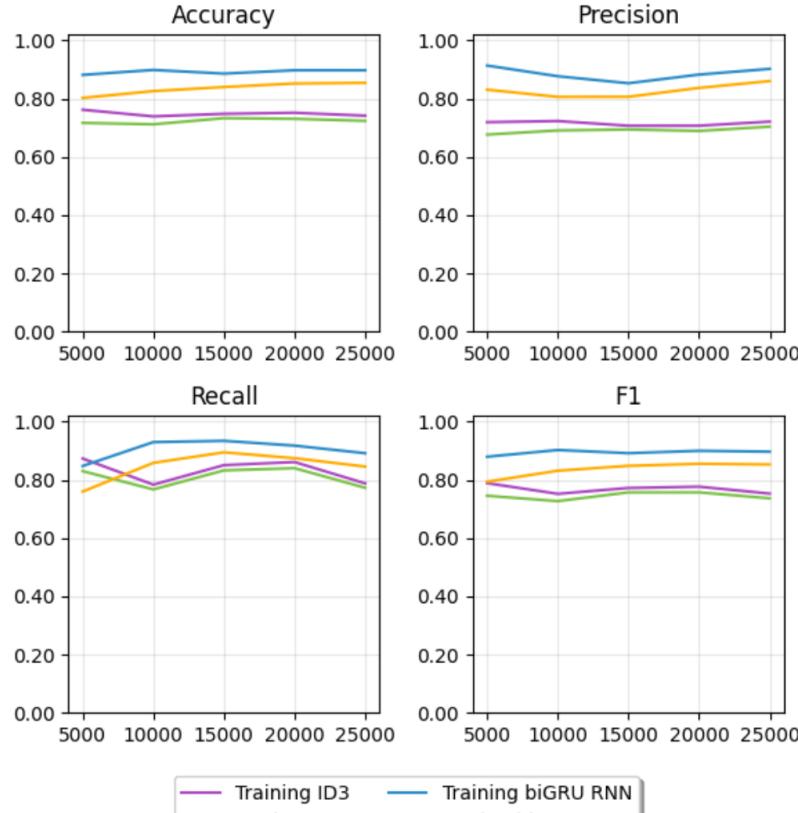
## Learning Curve Comparison for biGRU RNN against BernoulliNaiveBayes



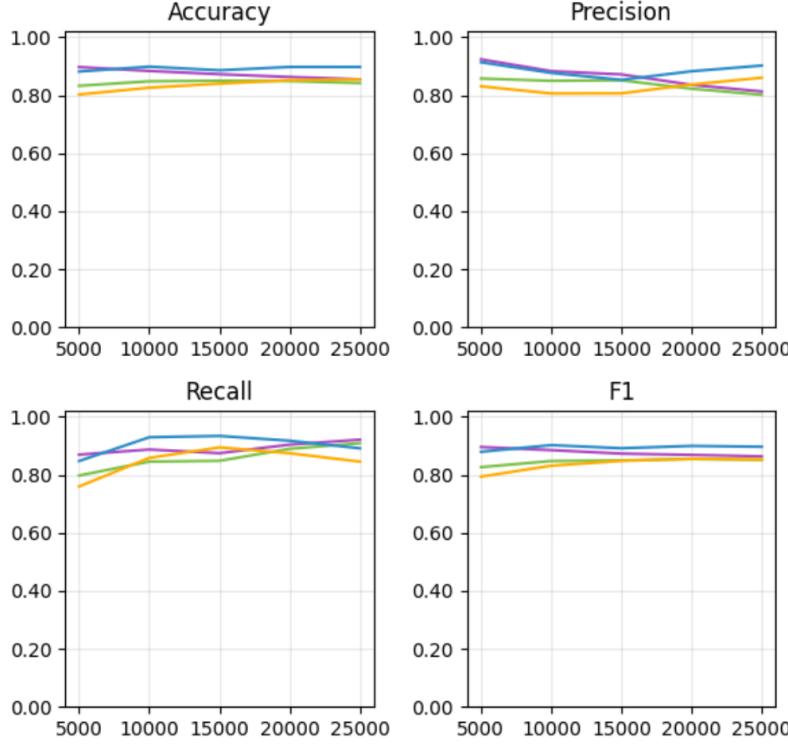
	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.88	0.80	0.91	0.83	0.85	0.76	0.88	0.79
10000	0.90	0.83	0.88	0.81	0.93	0.86	0.90	0.83
15000	0.89	0.84	0.85	0.81	0.93	0.89	0.89	0.85
20000	0.90	0.85	0.88	0.84	0.92	0.87	0.90	0.86
25000	0.90	0.85	0.90	0.86	0.89	0.85	0.90	0.85

Classification Table Difference for biGRU RNN against BernoulliNaiveBayes								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	0.050000	-0.020000	0.090000	0.020000	0.000000	-0.080000	0.040000	-0.040000
10000	0.080000	0.010000	0.070000	0.010000	0.070000	0.020000	0.070000	0.010000
15000	0.070000	0.020000	0.040000	0.010000	0.080000	0.050000	0.060000	0.030000
20000	0.080000	0.030000	0.070000	0.030000	0.070000	0.030000	0.070000	0.030000
25000	0.080000	0.040000	0.100000	0.060000	0.040000	0.010000	0.080000	0.030000

## Learning Curve Comparison for biGRU RNN against ID3



## Learning Curve Comparison for biGRU RNN against LogisticRegression



	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.120000	0.080000	0.190000	0.150000	-0.020000	-0.070000	0.090000	0.040000
10000	0.160000	0.120000	0.160000	0.120000	0.150000	0.090000	0.150000	0.100000
15000	0.140000	0.110000	0.140000	0.120000	0.080000	0.060000	0.120000	0.090000
20000	0.150000	0.120000	0.170000	0.150000	0.060000	0.030000	0.120000	0.100000
25000	0.160000	0.130000	0.180000	0.160000	0.100000	0.080000	0.150000	0.110000

Classification Table Difference for biGRU RNN against LogisticRegression								
Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test	
5000	-0.020000	-0.030000	-0.010000	-0.030000	-0.020000	-0.040000	-0.020000	-0.040000
10000	0.020000	-0.020000	0.000000	-0.040000	0.040000	0.010000	0.010000	-0.020000
15000	0.020000	-0.010000	-0.020000	-0.040000	0.060000	0.040000	0.020000	0.000000
20000	0.040000	0.000000	0.040000	0.020000	0.020000	-0.020000	0.030000	0.000000
25000	0.050000	0.010000	0.090000	0.060000	-0.030000	-0.060000	0.040000	0.000000