Τεχνητη Νοημοσύνη 2η εργασία

Μαθητές: Νικόλαος Ρώσσης (3200168) ,Γεράσιμος Ποταμιάνος (3200160)

k, m, n Υπερπαράμετροι λέξεων

Ρίχοντας μία ματιά στο imdb dataset μπορούμε να δούμε ότι οι πρώτες λέξεις είναι τύπου the, of, and, what οι οποίες δεν έχουν εννοιολογικά μία αρνητική ή θετική χροιά. Επομένως επιλέγουμε να ξεκινήσουμε μόλις βρούμε τις πρώτες σημαντικές λέξεις όπως το “movie” ή το “good’’ οι οποίες είναι περίπου μετά τις πρώτες 200 λέξεις.
(skip_top = 200, δηλαδή η υπερπαραμ. “n”)
Επειδή το dataset περιέχει ένα τεράστιο λεξιλόγιο η υπερπαράμετρος “k” εννοείται καθώς επιλέγουμε το num_words (υπερπαράμετρος “m”)
δηλαδη: k = Λεξεις dataset – num_words

Ολες οι συγκρίσεις και ο κώδικας που έχει παραχθεί γίνονται με 3800 λέξεις (200 πρώτες skipped και 4000 ως num_words)

Παράδειγμα με 500 Λέξεις (στον Naïve Bayes)

25000

	precision	recall	f1-score	support
0	0.80	0.77	0.78	12500
1	0.78	0.81	0.79	12500
accuracy			0.79	25000
macro avg	0.79	0.79	0.79	25000
weighted avg	0.79	0.79	0.79	25000

Παράδειγμα με 4000 Λέξεις (στον Naïve Bayes)

```
from sklearn.metrics import classification_report

classifier = NaiveBayes(laplace = 1.0)
classifier.fit(x_train_binary, y_train)

print(classification_report(y_train, classifier.predict(x_train_binary),
                           zero_division=1))
print(classification_report(y_test, classifier.predict(x_test_binary),
                           zero_division=1))
```

25000

	precision	recall	f1-score	support
0	0.86	0.87	0.86	12500
1	0.87	0.86	0.86	12500
accuracy			0.86	25000
macro avg	0.86	0.86	0.86	25000
weighted avg	0.86	0.86	0.86	25000

	precision	recall	f1-score	support
0	0.84	0.86	0.85	12500
1	0.86	0.83	0.84	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Μέρος Α – Ανάλυση Αποτελεσμάτων

```
from sklearn.metrics import classification_report

classifier = NaiveBayes(laplace = 1.0)
classifier.fit(x_train_binary, y_train)

print(classification_report(y_train, classifier.predict(x_train_binary),
                           zero_division=1))
print(classification_report(y_test, classifier.predict(x_test_binary),
                           zero_division=1))
```

```
25000
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     12500
           1       0.87      0.86      0.86     12500

    accuracy                           0.86     25000
   macro avg       0.86      0.86      0.86     25000
weighted avg       0.86      0.86      0.86     25000

              precision    recall  f1-score   support

           0       0.84      0.86      0.85     12500
           1       0.86      0.83      0.84     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000
```

Naïve Bayes

Όπως βλέπουμε το μοντέλο μας δεν είναι overfitting καθώς το accuracy όπως και οι υπόλοιποι μετρητές φαίνεται να έχουν μικρή απόκλιση (0.01 – 0.02) μεταξύ του test και του train set.

Το μοντέλο μας φαίνεται να λειτουργεί αρκετά καλά καθώς έχει accuracy 85% σε ξένα δεδομένα. Βρίσκει σε πολύ ικανοποιητικό βαθμό τις σωστές κλάσεις για τα δεδομένα (precision και recall) και επίσης οι δύο κλάσεις δεν έχουν σοβαρή απόκλιση ως προς αυτούς τους μετρητές (0.2-0.3). Την αναλογία αυτή μας δείχνει πιο ξεκάθαρα και το f1-score το οποίο είναι 0.84 και 0.85, αρκετά κοντά στο 1 και δείχνει μια ισορροπημένη σχέση ανάμεσα στο recall και το precision για την κάθε κλάση αντίστοιχα

```
classifier = LogisticRegression(0.001, 0.1, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.85     12500
           1       0.86      0.85      0.85     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000

              precision    recall  f1-score   support

           0       0.84      0.85      0.85     12500
           1       0.85      0.84      0.84     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000

✓ 1m 26s  completed at 4:49 PM
```

Logistic Regression

Δοκιμάστηκαν αρκετές τιμές αλλά αυτές φαίνονται να είναι οι optimal h (learning rate) = 0.001, lambda = 0.1, repeats = 100.
Όπως βλέπουμε το μοντέλο μας δεν είναι overfitting καθώς το accuracy όπως και οι υπόλοιποι μετρητές φαίνεται να έχουν ελάχιστη διαφορά (0.01) μεταξύ του test και του train set.
Το μοντέλο μας φαίνεται να λειτουργεί αρκετά καλά καθώς έχει accuracy 85% σε ξένα δεδομένα. Βρίσκει σε πολύ ικανοποιητικό βαθμό τις σωστές κλάσεις για τα δεδομένα (precision και recall) και επίσης οι δύο κλάσεις δεν έχουν σχεδόν καμία απόκλιση ως προς αυτούς τους μετρητές (0.01). Την αναλογία αυτή μας δείχνει πιο ξεκάθαρα και το f1-score το οποίο είναι 0.84 και 0.85, αρκετά κοντά στο 1 και δείχνει μια ισορροπημένη σχέση ανάμεσα στο recall και το precision για την κάθε κλάση αντίστοιχα

Μέρος Α - Επεξήγηση Υπερπαραμέτρων

Lamba >= 0.2

```
classifier = LogisticRegression(0.001, 0.2, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

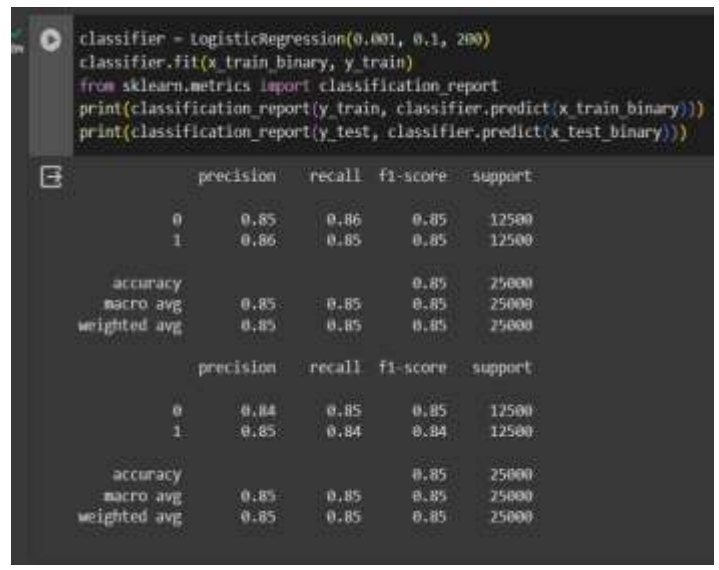
	precision	recall	f1-score	support
0	0.83	0.86	0.85	12500
1	0.86	0.82	0.84	12500
accuracy			0.84	25000
macro avg	0.84	0.84	0.84	25000
weighted avg	0.84	0.84	0.84	25000

	precision	recall	f1-score	support
0	0.82	0.87	0.84	12500
1	0.86	0.81	0.83	12500
accuracy			0.84	25000
macro avg	0.84	0.84	0.84	25000
weighted avg	0.84	0.84	0.84	25000

```
[77] from sklearn.model_selection import train_test_split
     # split rest of the train data into train and dev
```

1m 28s

Repeats > 100



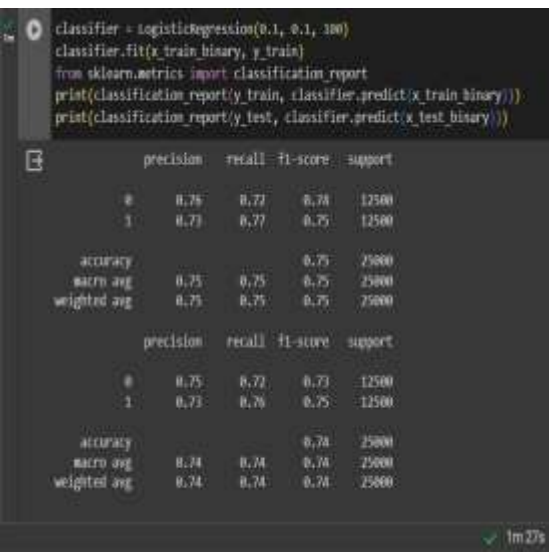
```
classifier = LogisticRegression(0.001, 0.1, 200)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

	precision	recall	f1-score	support
0	0.85	0.86	0.85	12500
1	0.86	0.85	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

	precision	recall	f1-score	support
0	0.84	0.85	0.85	12500
1	0.85	0.84	0.84	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Θέτοντας το λαμβδα = 0.2 αρχικά βλέπουμε ότι υπάρχει μεγαλύτερη διαφορά μεταξύ του train και του test classification report, επομένως δεν αποτελεί καλή τιμή για να αποφύγουμε overfitting. Επίσης, όχι μόνο μειώνει το accuracy (για 0.1) αλλά φαίνεται να επηρεάζει και το recall μεταξύ των δύο κλάσεων.
Με repeats μεγαλύτερα από 100 μπορούμε να δούμε ότι δεν υπάρχει αλλαγή (εκτός και αν βάλουμε μεγάλο h) επομένως δεν είναι σωστό trade-off να ανεβάσουμε τις επαναλήψεις με κόστος στον χρόνο (1 λεπτό ανά 100 επαναλήψεις).

h = 0.1



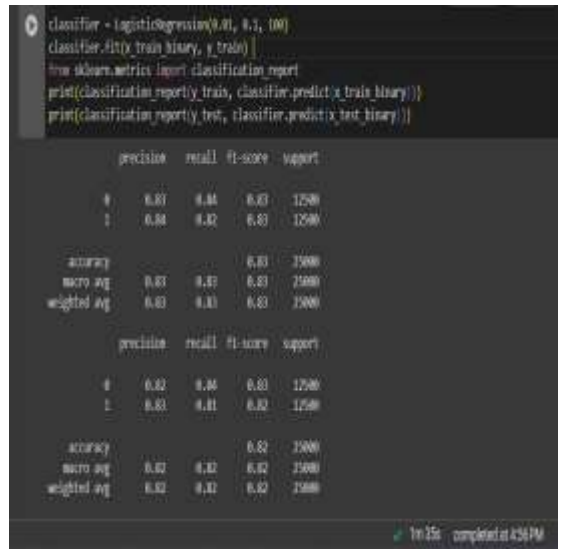
```
classifier = LogisticRegression(0.1, 0.1, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

	precision	recall	f1-score	support
0	0.76	0.72	0.74	12500
1	0.73	0.77	0.75	12500
accuracy			0.75	25000
macro avg	0.75	0.75	0.75	25000
weighted avg	0.75	0.75	0.75	25000

	precision	recall	f1-score	support
0	0.75	0.72	0.73	12500
1	0.73	0.76	0.75	12500
accuracy			0.74	25000
macro avg	0.74	0.74	0.74	25000
weighted avg	0.74	0.74	0.74	25000

1m 27s

h = 0.01



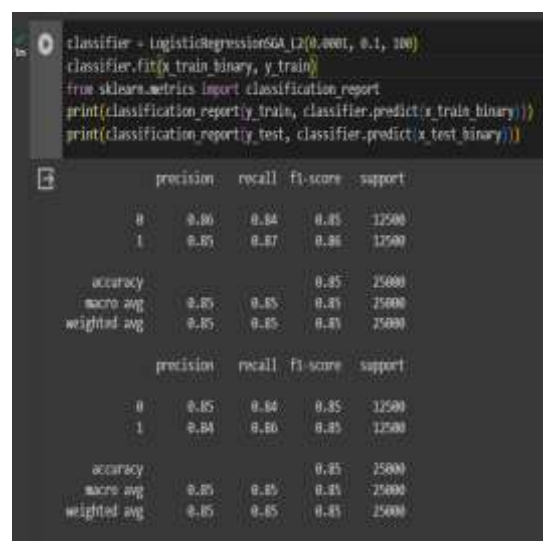
```
classifier = LogisticRegression(0.01, 0.1, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

	precision	recall	f1-score	support
0	0.83	0.84	0.83	12500
1	0.84	0.82	0.83	12500
accuracy			0.83	25000
macro avg	0.83	0.83	0.83	25000
weighted avg	0.83	0.83	0.83	25000

	precision	recall	f1-score	support
0	0.82	0.84	0.83	12500
1	0.83	0.81	0.82	12500
accuracy			0.82	25000
macro avg	0.82	0.82	0.82	25000
weighted avg	0.82	0.82	0.82	25000

1m 25s completed at 4:56 PM

h = 0.0001



```
classifier = LogisticRegressionSGA_12(0.0001, 0.1, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

	precision	recall	f1-score	support
0	0.86	0.84	0.85	12500
1	0.85	0.87	0.86	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

	precision	recall	f1-score	support
0	0.85	0.84	0.85	12500
1	0.84	0.86	0.85	12500
accuracy			0.85	25000
macro avg	0.85	0.85	0.85	25000
weighted avg	0.85	0.85	0.85	25000

Μπορούμε να δούμε ότι όλοι οι μετρητές στο 0.1 έχουν χαμηλότερες τιμές από τις optimal που επιλέξαμε (0.10 διαφορά). Οι τιμή 0.01 και αν και κοντά στο optimal classification report που έχουμε εξακολουθεί να αποτελεί υποδεέστερη επιλογή. Η 0.0001 φαίνεται να είναι ίδια με την 0.001 επομένως δεν έχει νόημα να ψάχνουμε μικρότερες τιμές.

Μέρος Β – Σύγκριση Αλγορίθμων με του Sklearn

Logistic Regression με SGA και L2 regul.

```
classifier = LogisticRegression(0.001, 0.1, 100)
classifier.fit(x_train_binary, y_train)
from sklearn.metrics import classification_report
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.85     12500
           1       0.86      0.85      0.85     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000

              precision    recall  f1-score   support

           0       0.84      0.85      0.85     12500
           1       0.85      0.84      0.84     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000
```

1m 26s completed at 4:49 PM

SkLearn LogReg

Classification Report

```
from sklearn.linear_model import SGDClassifier
classifier = SGDClassifier(loss='log_loss', learning_rate='constant', eta0=0.001,
                           alpha=0.1, max_iter=100)
classifier.fit(x_train_binary, y_train)
print(classification_report(y_train, classifier.predict(x_train_binary)))
print(classification_report(y_test, classifier.predict(x_test_binary)))
```

```
              precision    recall  f1-score   support

           0       0.88      0.82      0.85     12500
           1       0.83      0.89      0.86     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000

              precision    recall  f1-score   support

           0       0.87      0.81      0.84     12500
           1       0.82      0.88      0.85     12500

    accuracy                           0.84     25000
   macro avg       0.84      0.84      0.84     25000
weighted avg       0.84      0.84      0.84     25000
```

Παρατηρούμε ότι η δική μας υλοποίηση είναι λίγο καλύτερη από αυτή του sklearn καθώς έχει κατά 0.1 καλύτερο accuracy όπως και macro και weighted avg. Τα υπόλοιπα φαίνονται ίδια με τη διαφορά ότι το f1-score του sklearn είναι κατά 0.1 καλύτερο στη κλάση 1, ενώ στη δική μας υλοποίηση το ανάποδο.

Bernoulli Naïve Bayes

```
from sklearn.metrics import classification_report

classifier = NaiveBayes(laplace = 1.0)
classifier.fit(x_train_binary, y_train)

print(classification_report(y_train, classifier.predict(x_train_binary),
                           zero_division=1))
print(classification_report(y_test, classifier.predict(x_test_binary),
                           zero_division=1))
```

```
25000
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     12500
           1       0.87      0.86      0.86     12500

    accuracy                           0.86     25000
   macro avg       0.86      0.86      0.86     25000
weighted avg       0.86      0.86      0.86     25000

              precision    recall  f1-score   support

           0       0.84      0.86      0.85     12500
           1       0.86      0.83      0.84     12500

    accuracy                           0.85     25000
   macro avg       0.85      0.85      0.85     25000
weighted avg       0.85      0.85      0.85     25000
```

Sklearn BernoulliNB

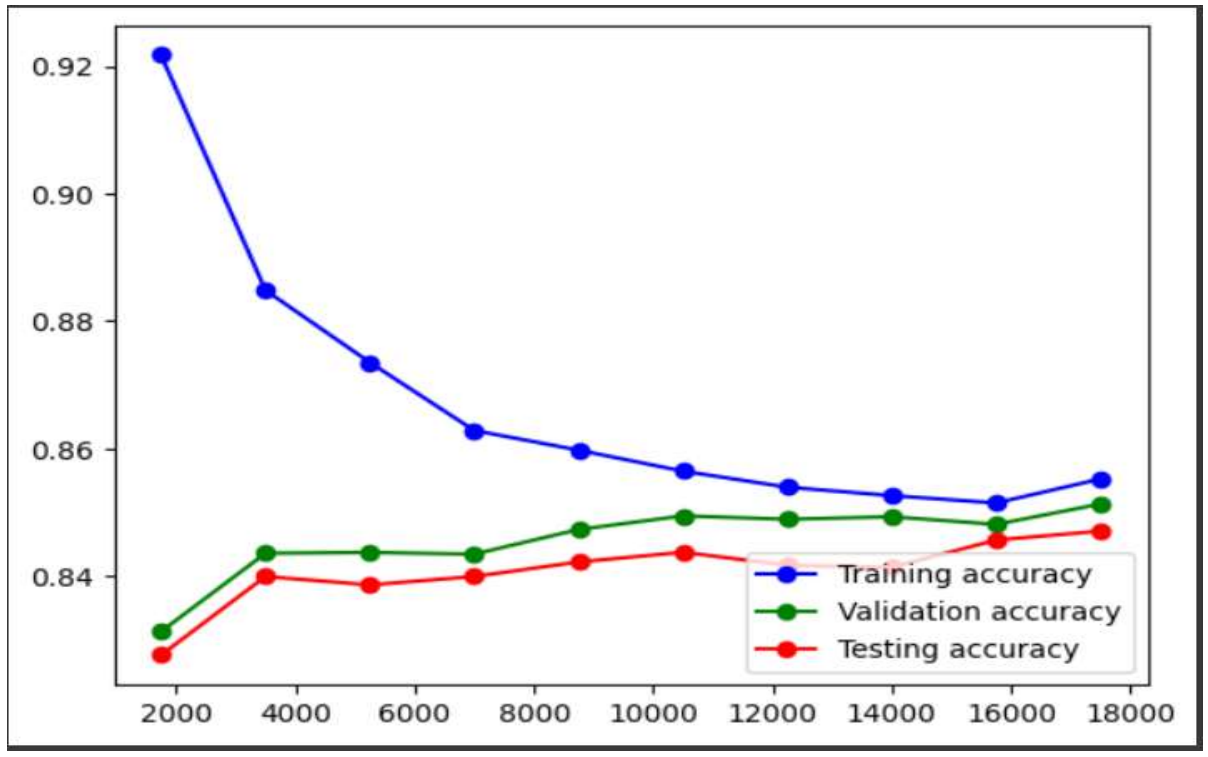
```
from sklearn.naive_bayes import BernoulliNB
bernoulli_nb = BernoulliNB(alpha = 1.0)
bernoulli_nb.fit(x_train_binary, y_train)
print(classification_report(y_train, bernoulli_nb.predict(x_train_binary),
                           zero_division=1))
print(classification_report(y_test, bernoulli_nb.predict(x_test_binary),
                           zero_division=1))
```

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86     12500
           1       0.87      0.85      0.86     12500

    accuracy                           0.86     25000
   macro avg       0.86      0.86      0.86     25000
weighted avg       0.86      0.86      0.86     25000

              precision    recall  f1-score   support

           0       0.83      0.86      0.84     12500
           1       0.86      0.82      0.84     12500

    accuracy                           0.84     25000
   macro avg       0.84      0.84      0.84     25000
weighted avg       0.84      0.84      0.84     25000
```

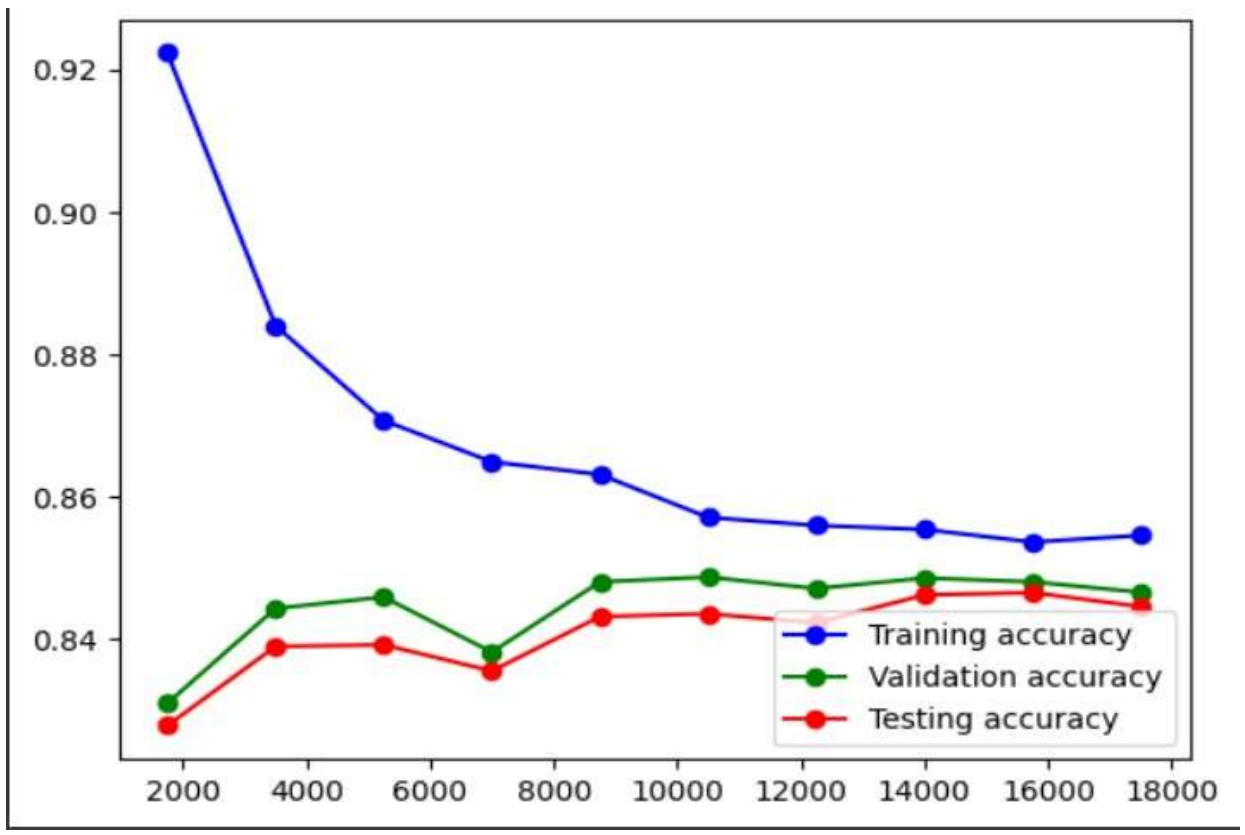
Παρατηρούμε ότι η δική μας υλοποίηση είναι λίγο καλύτερη από αυτή του sklearn γιατί έχει κατά 0.1 καλύτερο accuracy καθώς επίσης και καλύτερο f1 score και στις δύο κλάσεις (0.85 και περίπου 0.845) πράγμα που σημαίνει ότι έκανε πιο εύστοχο predict.

Αξιοσημείωτο είναι ότι επίσης ότι ενώ στο training data έχουν τα ίδια ακριβώς scores η δική μας υλοποίηση μπόρεσε να ανταπεξέλθει καλύτερα σε ξένα δεδομένα.

Sklearn Logistic Regression



Logistic Regression SGA με L2 reg.

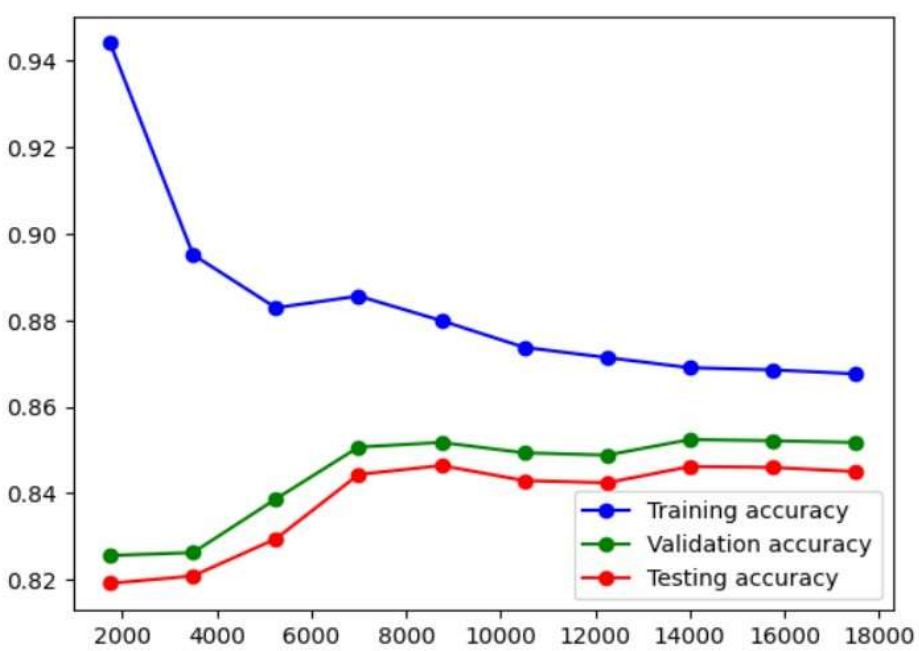


Οι δύο καμπύλες training φαίνεται να έχουν την ίδια πτώση ξεκινώντας λίγο πάνω απτό 0.92 και φτάνοντας περίπου στο 0.885. Έπειτα εμφανίζουν μία πτώση, της sklearn μεχρι το 0.86, ενώ η δική μας μέχρι το 0.865. Στη συνέχεια δείχνουν μία σταθερή πτώση μέχρι το 9^{ο} split. Η sklearn έχει μία εμφανή άνοδο στο 10^{ο} split κοντά στο 0.86 ενώ η δική μας μία πιο αχνή που επίσης φτάνει το 0.86.

Το validation και το testing του sklearn φαίνεται να ακολουθούν μία σχετικά ίδια πορεία με μικρή διαφορά (<0.1). Στο 7^{ο} και 8^{ο} split η testing φαίνεται λίγο κυρτή αλλά επιστρέφει μετά μαζί με την validation. Η validation σταθεροποιείται λίγο πιο πάνω απτο 0.85 ενώ η testing λίγο πιο κάτω απτό 0.85.

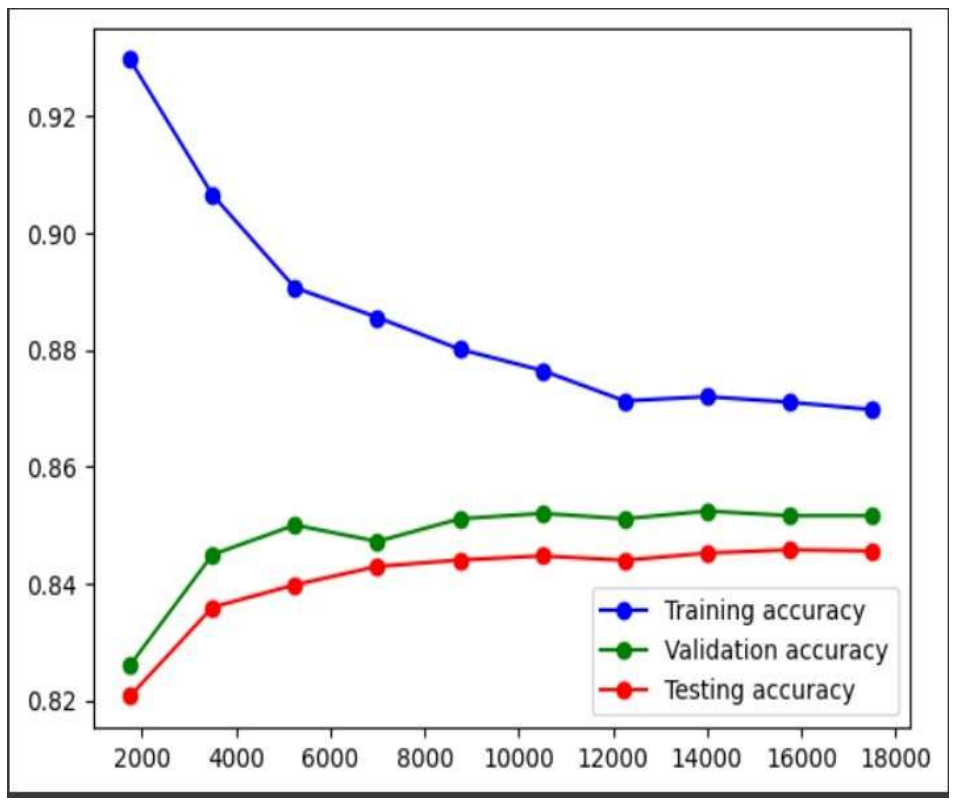
Οι δικές μας validation και testing ξεκινάνε όπως του sklearn αλλά δείχνουν απότομη άνοδο με τη validation να φτάνει το 0.85 ενώ του testing 0.84. Αμέσως μετά η validation δείχνει απότομη κάθοδο στο 0.84 και στη συνέχεια πάλι άνοδο όπου ακολουθεί ίδια πορεία με το training μέχρι να σταθεροποιηθούν και τα δύο στο 0.85

SkLearn Bern. Naïve Bayes



Η καμπύλη training του sklearn ξεκινάει από 0.94 και πέφτει απότομα στο 0.895. Συνεχίζει να ακολουθεί μια καθοδική πορεία (με ένα ίχνος ανόδου απτό περίπου 0.883 στο 0.889) πιο ομαλά μέχρι να σταθεροποιηθεί στο 0.87 (προς 0.86). Η καμπύλη του validation και του testing φαίνεται να ακολουθούν σχεδόν ίδια πορεία με διαφορά κατά περίπου 0.01. Το val ξεκινάει από 0.82 ενώ το testing από κάπου 0.825 έχουν και τα δύο μια σχετική άνοδο, με τη συνάρτηση να φαίνεται κυρτή σε αυτό το κομμάτι, κοντά στο 0.85 μέχρι που το validation σταθεροποιείται κοντά στο 0.85 και το testing κοντά στο 0.84.

Our Bernoulli Naïve Bayes

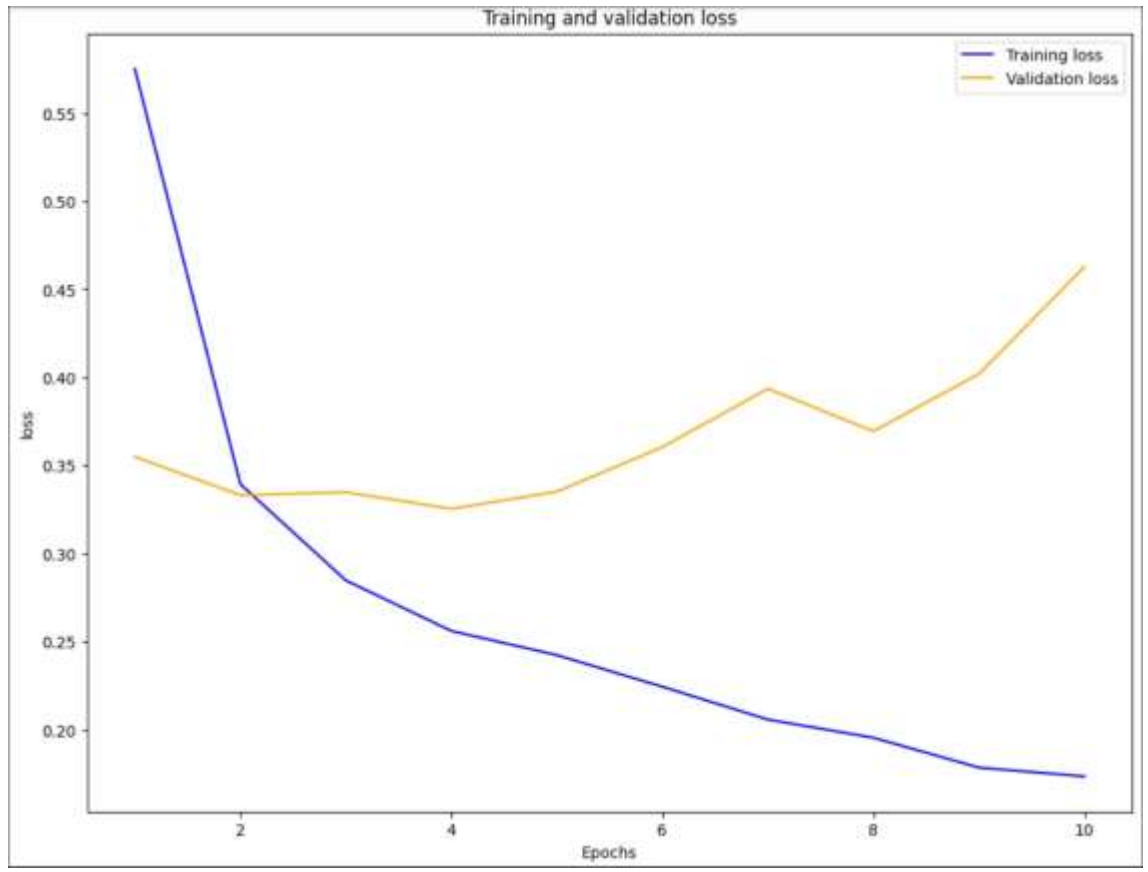


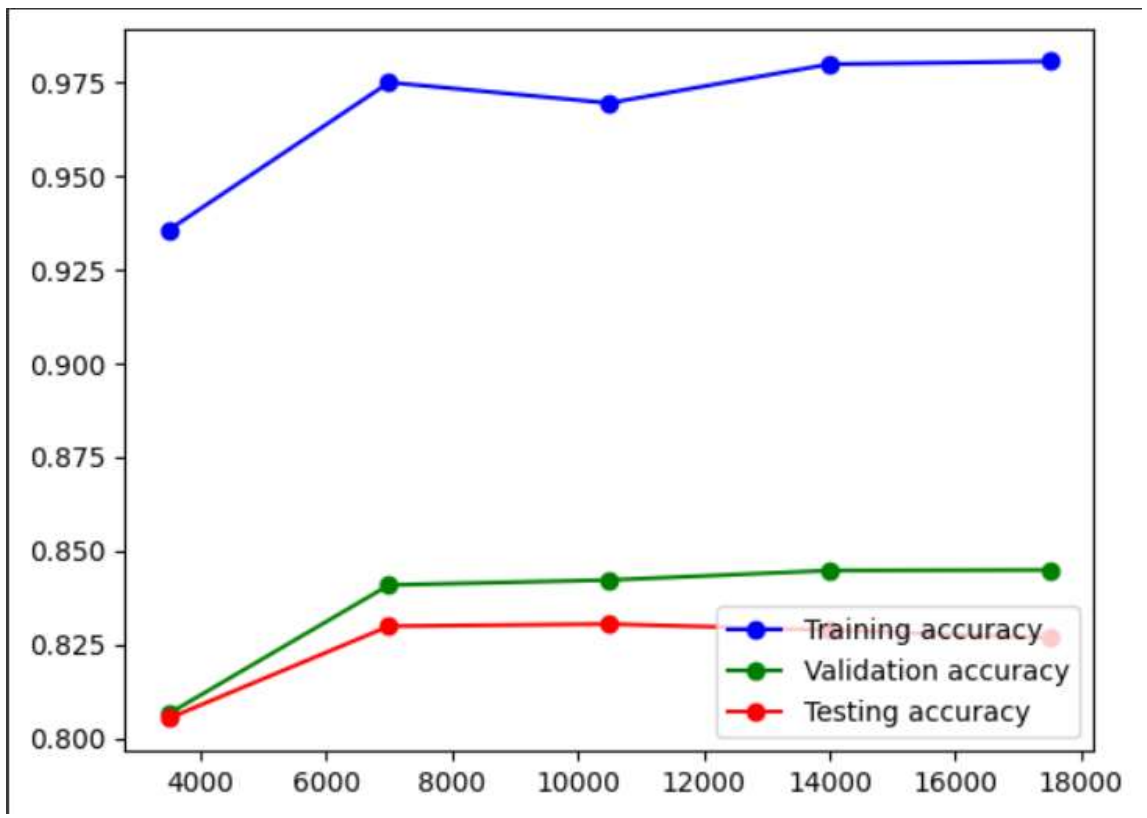
Η δική μας καμπύλη ξεκινάει απτό 0.93 αλλά έχει μια πιο ομαλή πτώση μέχρι το 0.89 και στη συνέχεια με ακόμα πιο σταθερή ομαλότητα πέφτει στο 0.87 (προς 0.86). Το validation και το testing ακολουθούν σχεδόν ίδια πορεία με το validation να εμφανίζει μία μικρή πτώση στο 4ο split. Το validation ξεκινάει από το 0.82, το testing από το 0.825, έχουν μία άνοδο, με τη συνάρτηση να φαίνεται κοίλη σε αυτό το κομμάτι, και σταθεροποιούνται και τα δύο κοντά στο 0.85.

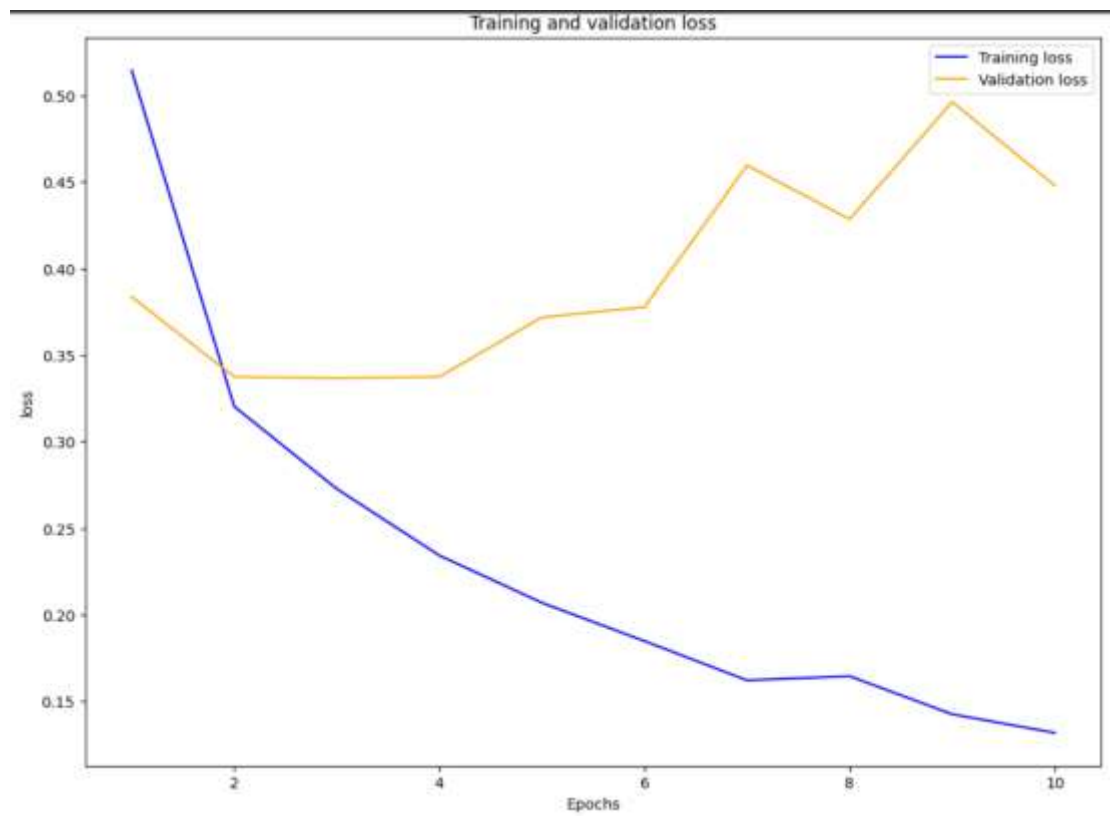
Μέρος Γ – RNN με Word Embeddings
Πίνακας – Loss Καμπύλη – Learning Curve

```
from sklearn.metrics import classification_report

y_pred_imdb = imdb_rnn.predict(x_test)
print(y_pred_imdb[0])

y_pred_imdb = (y_pred_imdb >= 0.5).astype(int)

print(classification_report(y_test, y_pred_imdb, zero_division=1))
```

```
782/782 [==============================] - 15s 17ms/step
[0.10328107]
              precision    recall  f1-score   support

           0       0.82      0.86      0.84     12500
           1       0.85      0.82      0.83     12500

    accuracy                           0.84     25000
   macro avg       0.84      0.84      0.84     25000
weighted avg       0.84      0.84      0.84     25000
```







Μέρος Γ – Σύγκριση Πινάκων, Καμπυλών

Τα αποτελέσματα του RNN φαίνεται να είναι ελάχιστα υποδεέστερα από αυτά των 2 άλλων αλγορίθμων του sklearn καθώς έχει το ίδιο accuracy και λίγο χαμηλότερο f1-score (0.01 χαμηλότερο από του sklearn) και κατ επέκτασην υποδεέστερα και απτά δικά μας που ήδη έχουμε συγκρίνει με του sklearn.

Η καμπύλη training φαίνεται να έχει μία διαρκώς ανοδική πορεία όσο μεγαλώνει ο αριθμός των εποχών. Το testing και το validation ξεκινάνε απτό ίδιο σημείο με το validation να ανεβαίνει ψηλότερα και στη συνέχεια ακολουθούν μία σταθερή πορεία μέχρι να σταθεροποιηθούν ανάμεσα στο 0.84. Συγκριτικά με τις δικές μας καμπύλες και αυτές του sklearn παρατηρούμε ότι το training έχει μονίμως ανοδική πορεία ενώ στις άλλες κατά κύριο λόγο καθοδική. Το validation και testing επίσης φαίνεται να μην έχουν πολλές εναλλαγές και ακολουθούν σταθερές πορείες. Εδώ ο χωρισμός των splits σε 10 είναι δύσκολος λόγω του χρόνου που χρειάζεται το RNN για fit και predicts

Μέρος Γ – Εξήγηση των Loss Plots

Κάναμε 2 Loss Plots για να καταννοήσουμε καλύτερα το πως λειτουργεί ένα RNN. Γενικότερα φαίνεται το training loss να πέφτει και το validation loss να ανεβαίνει όσο μεγαλώνουν οι εποχές. Αυτή η συμπεριφορά μπορεί να οφείλεται στο ότι δεν βάλαμε αρκετά κρυφά layers στο LSMT (16), στο ότι θα έπρεπε να ανεβάσουμε το dropout ή ενδεχομένως τα batches κατά το fit. Πάντως όπως δείχνει η καμπύλη μάθησης και ο πίνακας αποτελεσμάτων φαίνεται να κάνει σωστό binary classification.