DEPARTMENT OF INFORMATICS

ARTIFICIAL INTELIGENCE
WINTER SEMESTER 2023-2024

Eleni Kechrioti
Maria Schoinaki
Christos Stamoulos

Introduction

The current subject of this assignment is «**Sentiment analysis on movie reviews**» and aims at the classification of movie reviews to "negative" or "positive" using machine learning and neural networks algorithms.

The review dataset that we used is the **IMDB dataset of** Keras.

The **machine learning** algorithms we chose to implement are:

- **Random Forest**
- **Naive Bayes**
- **Logistic Regression**

The **neural network** we chose to implement is:

- **MLP**

Part A&B

Preprocessing data

With the assistance of tf.*keras.datasets.imdb.load_data ,* function*,* we extracted the review data and divided it into **training** data and **test** data . These data, of course, contained only the **m** most frequent words, minus the **n** most frequent and **k** rarest of them. This was achieved by putting as arguments to the function, m-k for num_words and n for skip_top. **m, n, k** are **hyperparameters**, which we defined for each different algorithm, to achieve the optimal result. Movie reviews are stored in **arrays x_train, x_test** and the category in which each film review belongs is defined in the arrays **y_train, y_test**, with each of these arrays having size of **25,000**. The arrays **y_train, y_test** contain **0** and **1,** with **1** representing a **positive review** and **0** representing a **negative review**. Next, we modified the arrays **x_train, x_test**, so that instead of containing a list of numbers for each review (*which each number corresponds to one word*), they contain strings of words.

Finally, to formalize our data according to the programming logic, we converted them into binary data. More specifically **, the arrays x_train, x_test**, went through a *binary CountVectorizer* to be converted into **binary vectors**.
Each review corresponds to one vector (*25,000 vectors for each array*)). Each vector has an amount value, the size of our vocabulary and each value of the vector, if it is 1 It means that the specific word of the vocabulary appears in the current review, on the contrary if it is 0, it means, that it does not appear. We end up with **(x_train_binary, y_train)** and **(x_test_binary, y_test).**

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

Information Gain

For the **Naive Bayes** and **Logistic Regression algorithms**, an Information Gain *algorithm* was implemented, using **our lab's aids**, which examines information gain, but we decided not to use it as it consumed too much computational costs and did not offer satisfactory results. Specifically, the difference between the use and non-use was infinitesimal, so it was not finally profitable to apply the above algorithm.

Note: Throughout the work, the python's NumPy library was mostly used, which sped up the process as it reduced the computational load of our programs.

Naive Bayes

We implemented the **Bernoulli** form of the **Naive Bayes algorithm with** a Laplace **estimator** to avoid zero probabilities and use logarithmic probabilities to eliminate the error scenario due to excessively small multiplications.

Methods of the BernoulliNaiveBayes class

Fit

It is the method that we train the algorithm on a training dataset (x) with the correct responses (y).

Let us analyze further.

Firstly, we calculated the a-priori probabilities of each class, and we applied the logarithmic function on them. Next, we calculated the probability that an attribute is present (i.e., the attribute vector has a value of 1 for that attribute) given that a review belongs to a category. This information was stored in two variables, positive_1 and negative_1, with these probabilities being naturally logarithmic. Respectively, in two other variables, positive_0 and negative_0, we calculated the logarithmic probability in which an attribute will not occur (i.e. the attribute vector has a value of 0 for that attribute) since an example without this attribute belongs to each class.

The probabilities mentioned are calculated as follows:

1. We sum each column of the array with the training examples, and we end up with a vector that contains the number of times each feature occurred in a particular category.

1. We convert each value of the vector from sum to probability of occurrence for each attribute given by a class, by dividing the result of step 1 with the total quantity of the class with the application of Laplace.

Predict Its the method that predicts/categorizes a set of development given data (x).

On two variables positive and negative, we calculate for each review the probability that it will be positive or negative, given its features.
For this calculation, we need an auxiliary matrix x_reverse, which is the opposite of matrix x, so that it has 1 where x has 0 and vice versa. It will be needed for the inner product.
More specifically, for the positive category we calculate the dot product (inner product) with table x_reverse and matrix positive_0, as well as the matrix x with the positive_1 and then we add them up, along with the a-priori probability of the positive category. We do this because, the probability that a text belongs to the positive class, given its features, will be the sum of the probabilities that it belongs to the positive class based on the existence or not of every feature. As well as without thinking about the features, the general possibility that a review belongs to the positive class. Similarly for the negative category. Finally, we decide whether the text belongs to the positive or negative category depending on which probability is greater.

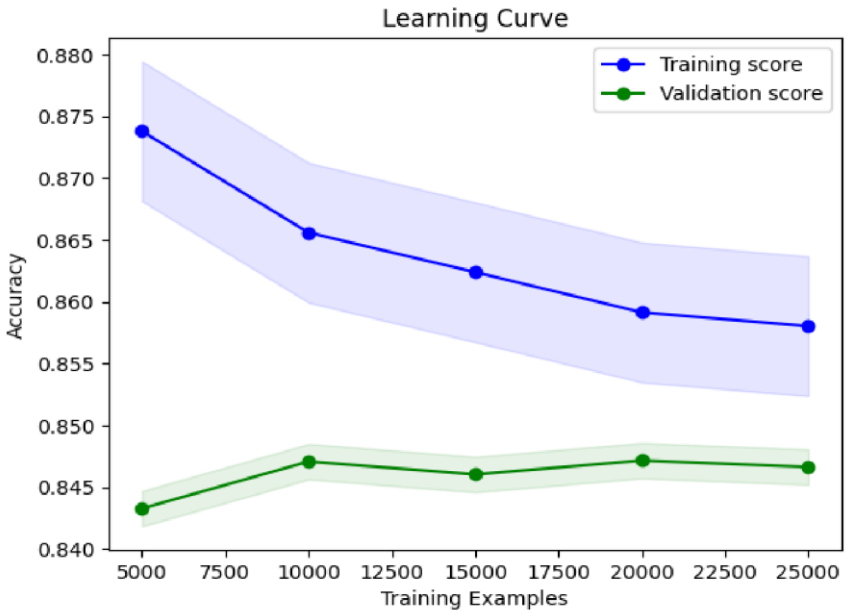
Results, Comparisons and Conclusions

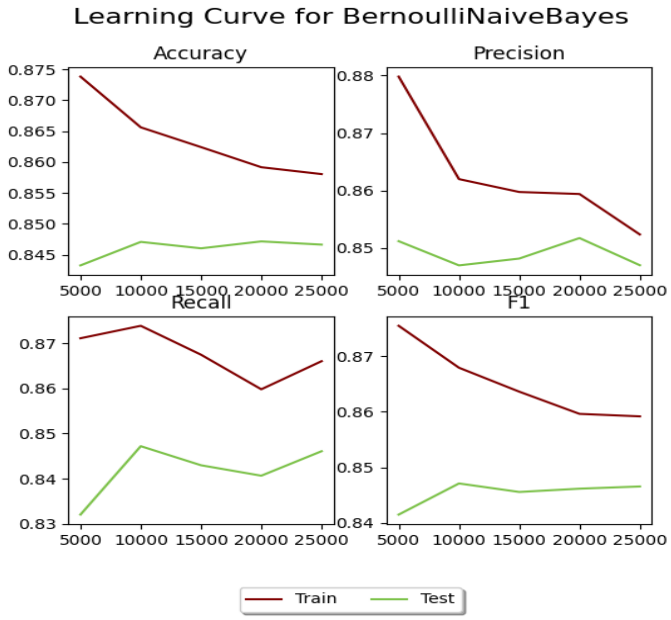
The results we analyze were made from experiments that had the following hyperparameters **m=3000, n=50, k=80, IG=false**. These hyperparameters were found after experimenting with various combinations of values and selecting the combination with the greatest accuracy.
As shown in the diagrams with the results of the **Naive Bayes algorithm**, the accuracy in train data and test data is remarkably close. As we increase the training dataset, the algorithm's performance decreases in train data, while it increases for tests. So, our algorithm does learn and **train** since the concept of "experience" is now felt and even **responds** particularly well to evaluation data. By training and evaluating the algorithm of the SkLearn library, and after comparing it with our own algorithm, we noticed (*as shown below in the charts*) that the percentages are quite close. From both the diagrams and the arrays (*and/or better the difference table*), the approaches are almost identical.

Πίνακας Διαφοράς για τον BernoulliNaiveBayes και τον SKLearn's BernoulliNB

	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	0.000000
10000	0.000000	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	0.000000
20000	0.000000	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	0.000000

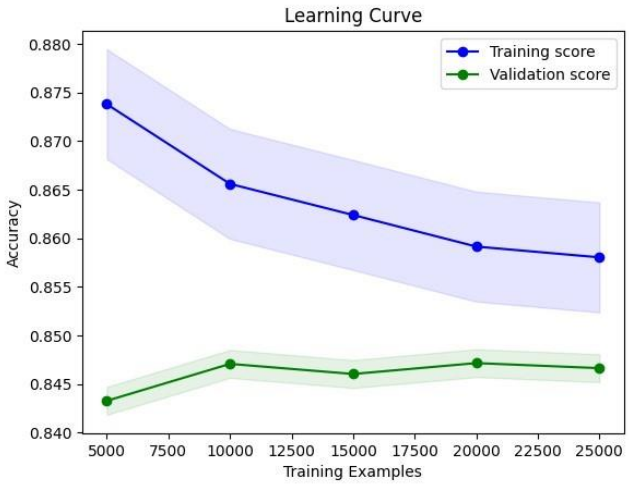
Learning, F1, Precision, Recall, F1 Curves for Naive Bayes

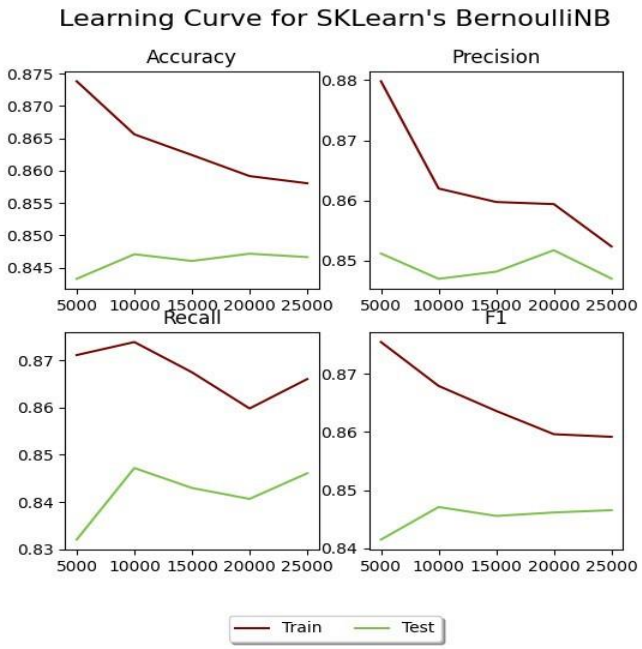


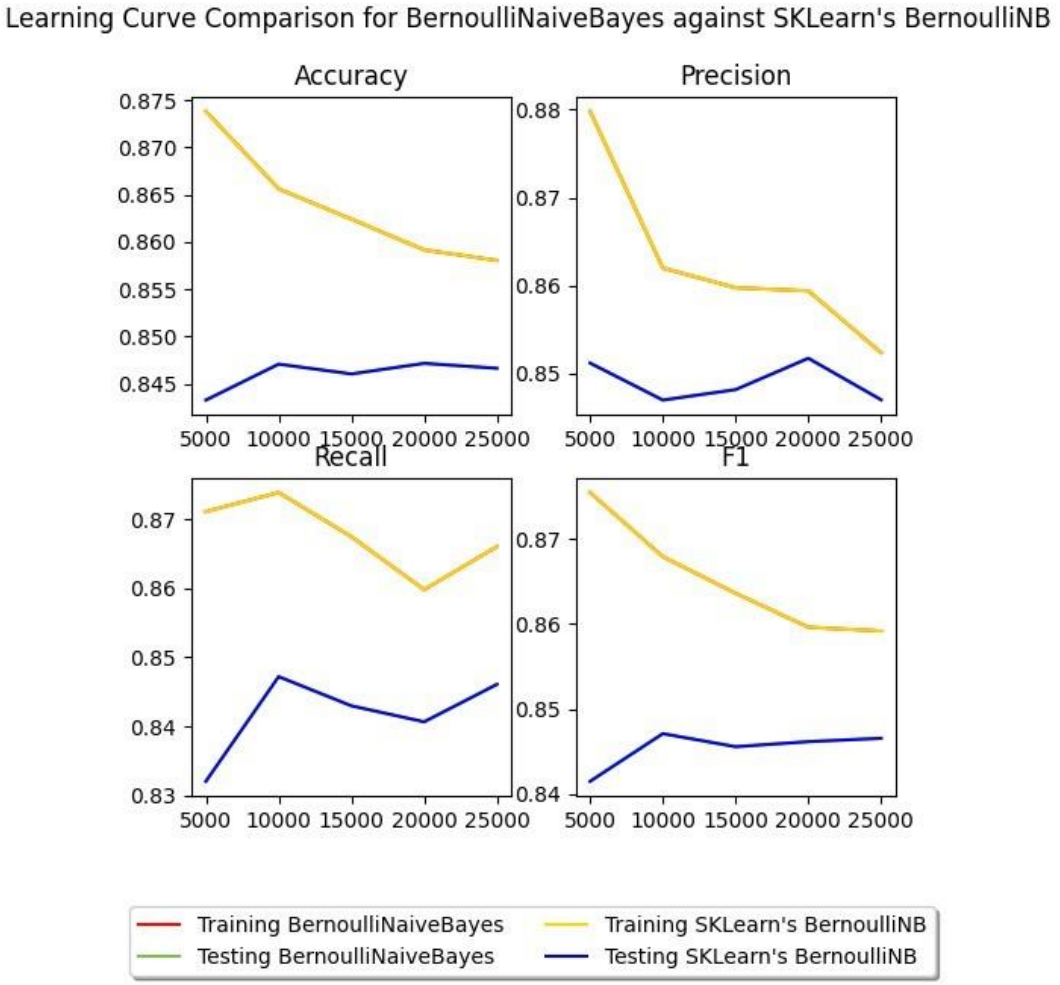


	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.87	0.84	0.88	0.85	0.87	0.83	0.88	0.84
10000	0.87	0.85	0.86	0.85	0.87	0.85	0.87	0.85
15000	0.86	0.85	0.86	0.85	0.87	0.84	0.86	0.85
20000	0.86	0.85	0.86	0.85	0.86	0.84	0.86	0.85
25000	0.86	0.85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

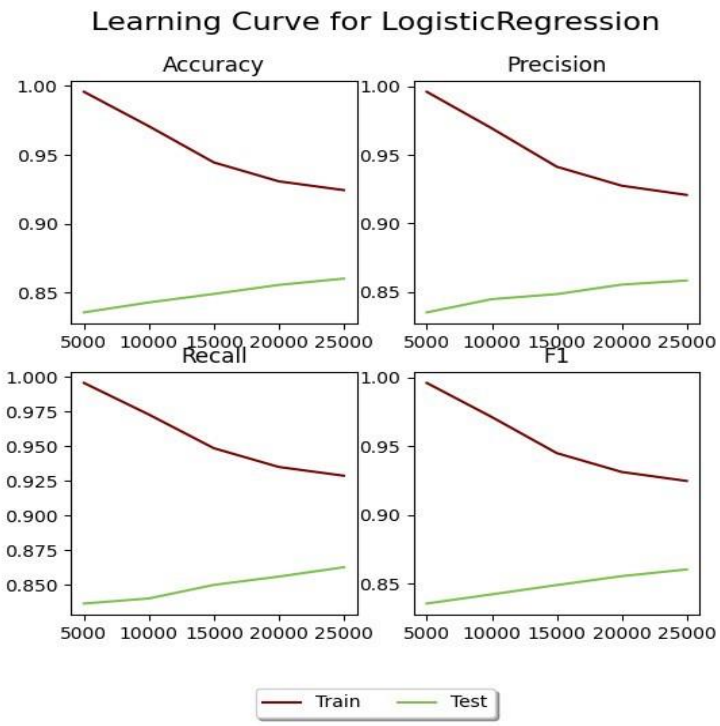
Learning, F1, Precision, Recall Curves for SkLearn's Naive Bayes

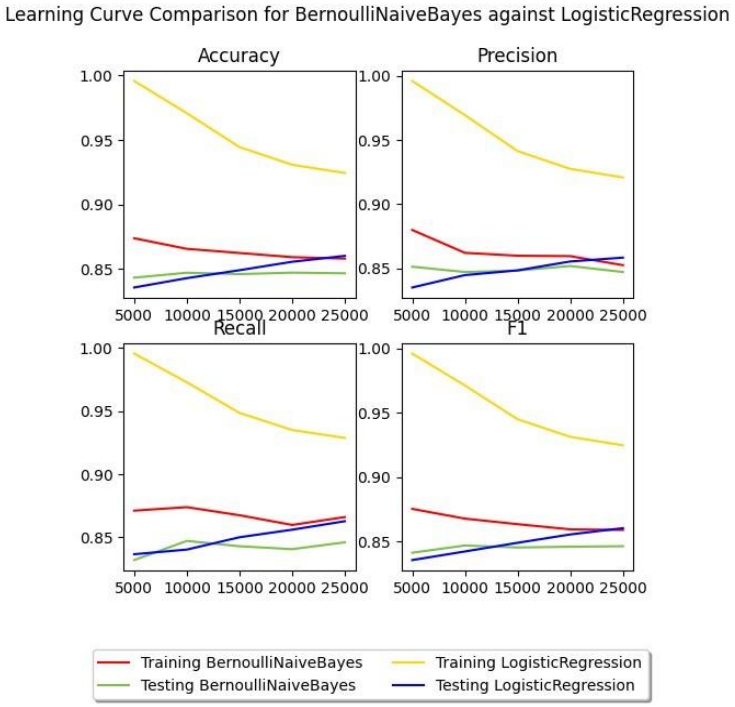






Learning, F1, Precision, Recall Curves for SkLearn's Logistic Regression





Πίνακας Διαφοράς για τον BernoulliNaiveBayes και τον LogisticRegression

	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.130000	0.000000	0.120000	0.020000	0.130000	0.010000	0.120000	0.000000
10000	0.100000	0.010000	0.110000	0.010000	0.100000	0.010000	0.100000	0.010000
15000	0.080000	0.000000	0.080000	0.000000	0.080000	0.010000	0.080000	0.000000
20000	0.070000	0.010000	0.070000	0.010000	0.080000	0.020000	0.070000	0.010000
25000	0.060000	0.010000	0.070000	0.010000	0.060000	0.010000	0.060000	0.010000

Logistic Regression

We implemented Logistic Regression with **lasso** normalization to avoid overfitting to the training data and gradient ascent.
Below we discuss the methods of the Logistic Regression class.

Fit:

It's the method by which we train the algorithm on a training dataset (x) with their correct responses (y).
We initialize the weights of the features with 0 and start the training of the algorithm which takes place in 3 levels. Every single level is repeated by the given maximum number of iterations (max_iter).

Specifically, the following levels are :

We calculate the inner product of weights with the matrix X and based on the result we calculate the predicted responses y_pred through the sigmoid function. We implement the gradient accent, having in mind an imaginary hill, to climb it to the top and find the global maxima.
We calculate the average of each vector of the inner product with the inverse matrix X and the difference of responses (y_pred-y). We normalize the ascent by subtracting from (a) the product of **λ** by the sum squares of weights (L2 normalization).
We sum up the weights of the stochastic gradient multiplied by **η**, to approach convergence at a uniform rate.

predict:

It's the method that predicts/categorizes a set of given development data (x).
The prediction is implemented as follows.
We calculate the inner product of the data given for prediction with the weights of the features stored after training. Next, we find the predicted responses through the sigmoid function and decide whether the final prediction belongs to the positive or negative category based on the fact that the sigmoid value is greater or lower than the constant threshold of 0.5.
The parameters lambda_value, eta and max_iter were appropriately selected through an experimentation algorithm. This special algorithm was implemented based on the accuracy score on development data with various candidate values. The chosen combination is lambda value: 0.01 , max_iterr: 1000 and eta: 0.001.

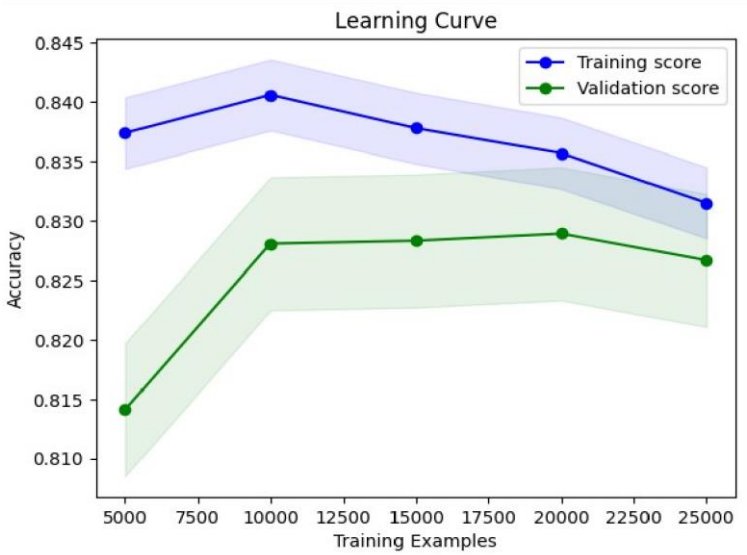
Results, Comparisons and Conclusions

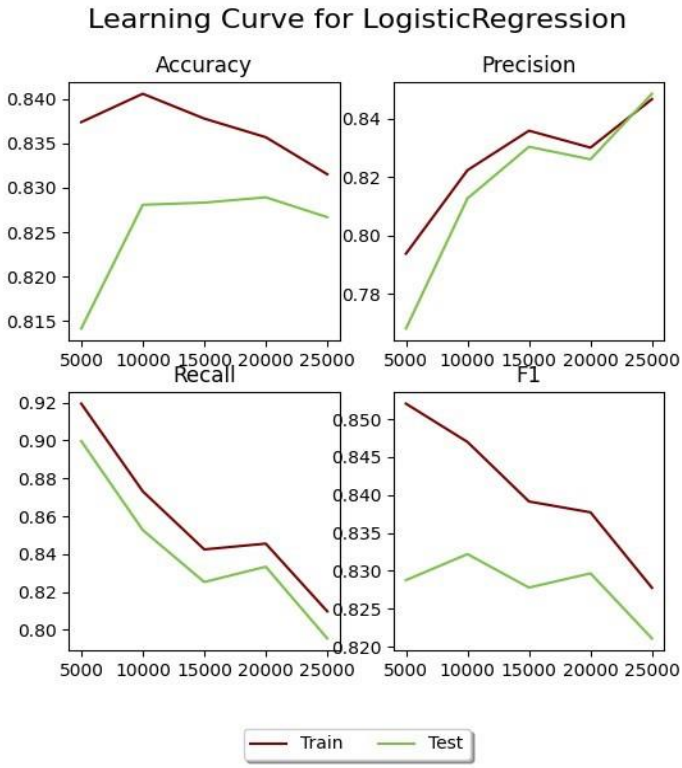
The results we analyze were made from experiments that had the following super parameters **m=2500, n=200, k=20, IG=false**. These hyperparameters were found after experimenting with various combinations of values and selecting this combination with the greatest accuracy. As shown in the diagrams with the results of the **Logistic Regration** algorithm, the accuracy in train data and test data is very close. As we increase the training dataset, the algorithm's performance on test data increases. So, our algorithm does learn and train, since the concept of "experience" is now felt and in fact responds particularly well to evaluation data. By training and evaluating the algorithm of the SkLearn library, and after comparing it with our own algorithm, we noticed (as shown below in the charts) that the percentages are quite close. From both the diagrams as well as the arrays (and/or better the difference table), the approximations are almost identical.

Classification Table Difference for LogisticRegression against SKLogisticRegression

	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.170000	0.030000	0.230000	0.090000	0.040000	0.100000	0.150000	0.000000
10000	0.120000	0.010000	0.150000	0.030000	0.070000	0.030000	0.110000	0.000000
15000	0.090000	0.010000	0.100000	0.020000	0.090000	0.000000	0.090000	0.010000
20000	0.080000	0.020000	0.100000	0.030000	0.070000	0.000000	0.080000	0.020000
25000	0.080000	0.020000	0.060000	0.010000	0.110000	0.060000	0.080000	0.030000

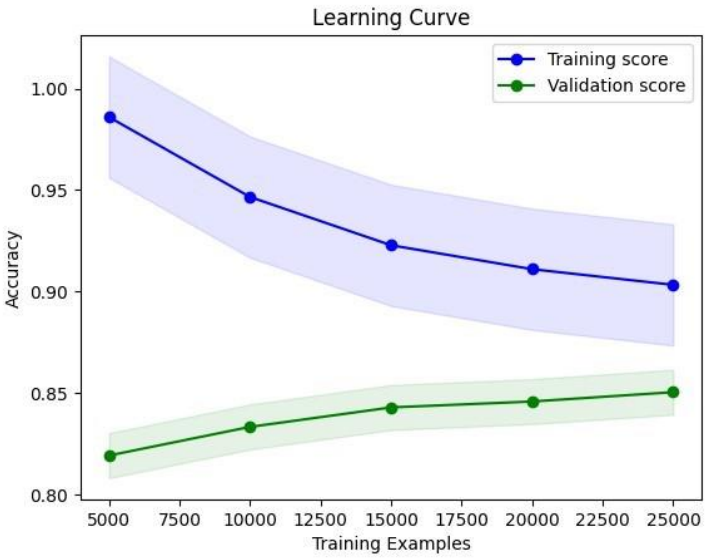
Learning, F1, Precision, Recall, F1 Curves for Logistic Regression

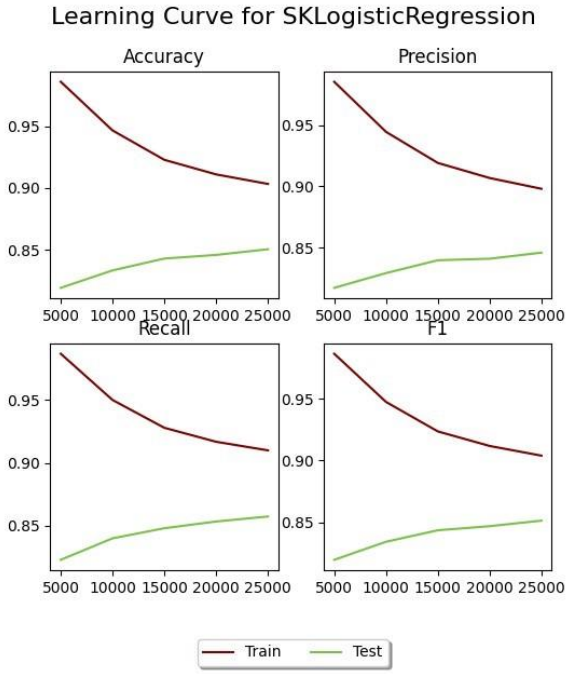


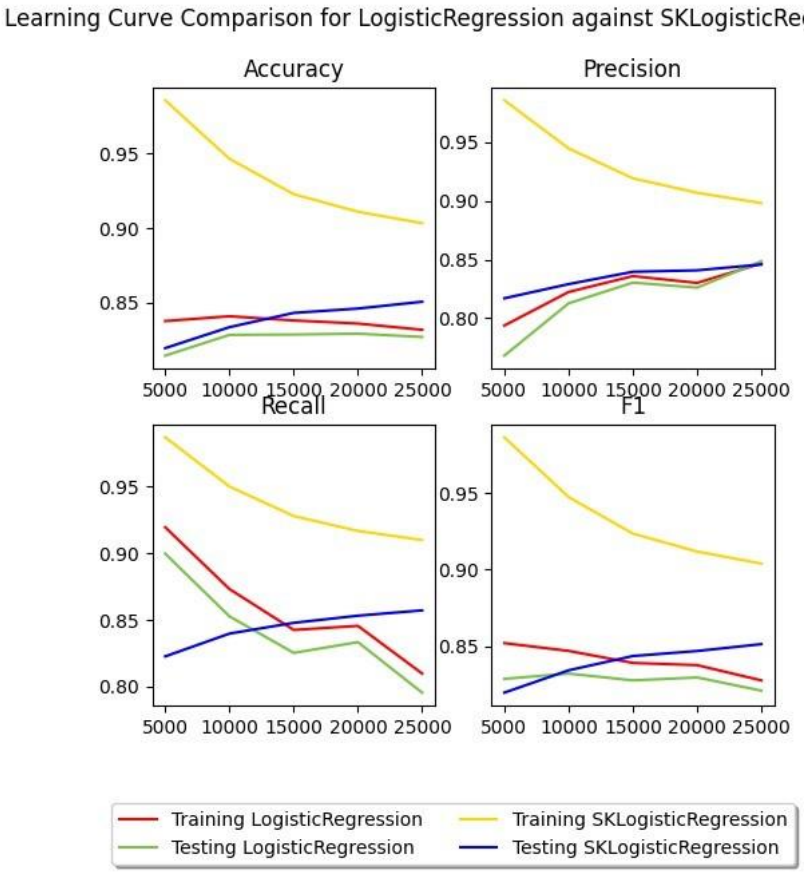


	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.84	0.81	0.79	0.77	0.92	0.90	0.85	0.83
10000	0.84	0.83	0.82	0.81	0.87	0.85	0.85	0.83
15000	0.84	0.83	0.84	0.83	0.84	0.83	0.84	0.83
20000	0.84	0.83	0.83	0.83	0.85	0.83	0.84	0.83
25000	0.83	0.83	0.85	0.85	0.81	0.80	0.83	0.82

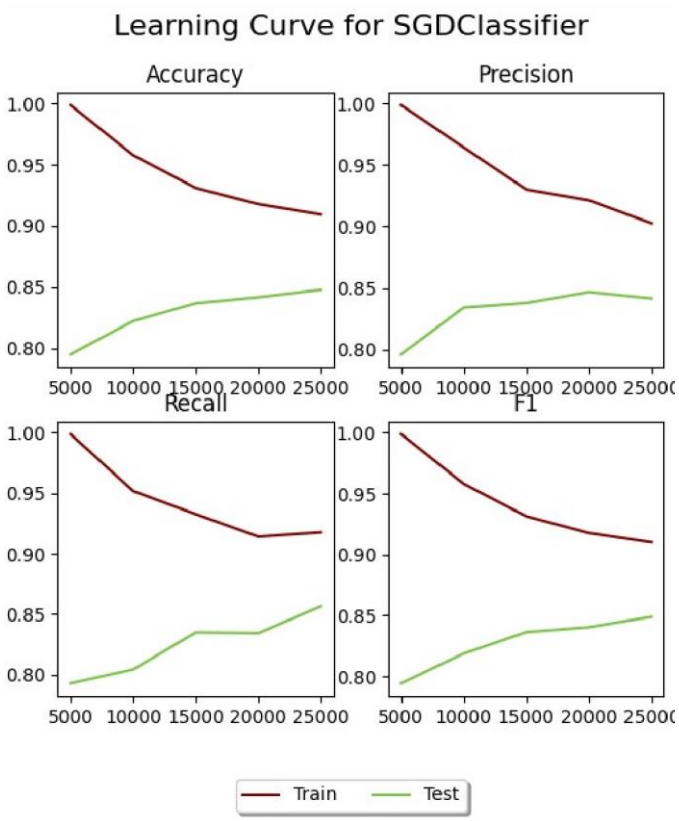
Learning, F1, Precision, Recall Curves for SkLearn's Logistic Regression

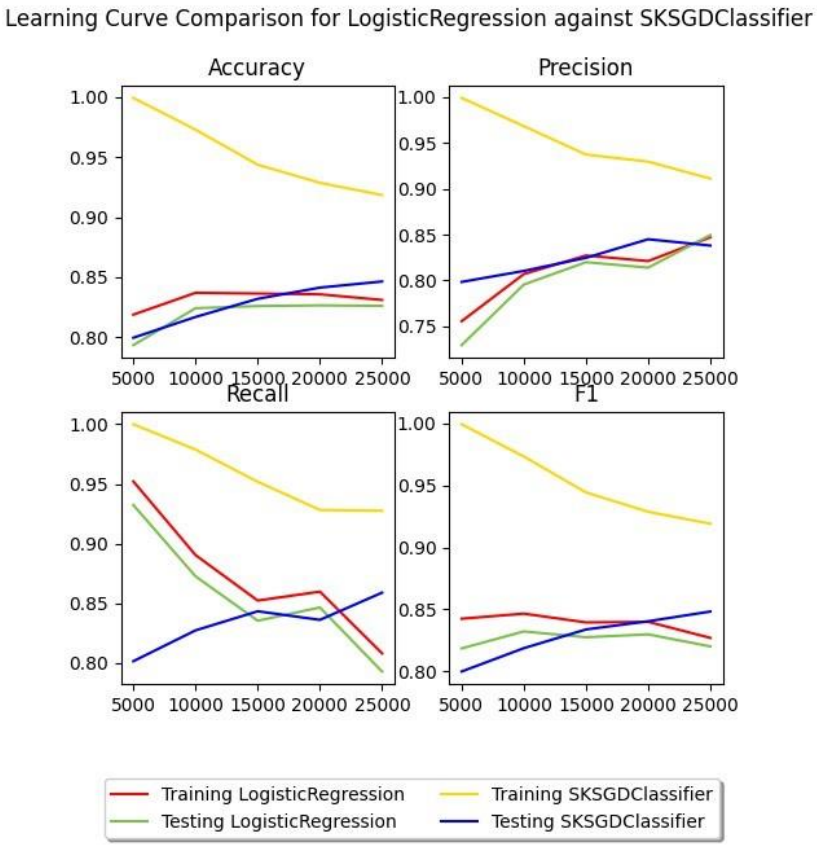






Learning, F1, Precision, Recall Curves for SkLearn's SGDClassifier





Classification Table Difference for LogisticRegression against SGDClassifier

	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.180000	0.010000	0.240000	0.080000	0.050000	0.140000	0.160000	0.020000
10000	0.130000	0.000000	0.170000	0.020000	0.080000	0.060000	0.120000	0.020000
15000	0.100000	0.000000	0.110000	0.010000	0.100000	0.000000	0.100000	0.000000
20000	0.090000	0.010000	0.110000	0.030000	0.070000	0.010000	0.090000	0.010000
25000	0.090000	0.020000	0.060000	0.020000	0.120000	0.080000	0.090000	0.030000

Random Forest

We implemented the **Random Forest algorithm**, which is a collective learning algorithm that implements the concept of majority among number_of_trees ID3 trees.

Random Forest methods:

Fit

It is the method in which **the algorithm is trained**. It starts by running a loop (*one for each tree*). In each iteration, it creates a new **ID3** tree and trains it with random examples of repositioning training and a random set with properties. To create these random sets, 2 new methods were implemented.

Select_random_samples

This method accepts the original data training dataset and produces a new one of the same size with random training examples, with duplicates. It also modifies the y table to have corresponding data and mapping.

Select_random_features

This method respectively takes as input the original data set of training data and selects at random m properties (super parameters) subset of the original vocabulary and without repositioning.

Predict

This method is the determinator of the algorithm. it decides and classifies each training example in the category in which the function believes every review belongs to. Initially for each tree, our function calls the predict method of the tree, giving the x test dataset as input. The ID3 predict returns a total of 0,1 which is the tree's responses to each evaluation example. Then the random forest predict runs for each evaluation example, every id3 prediction- response for that example. Whether it has a value of 1 or 0, the corresponding counter for 0 and 1 in that example increases. Once all id3 predictions have been examined for an example, the result for this example is stored in the final return array of Random Forest. So, each training example has the majority response of the random id3 trees created by random forest.

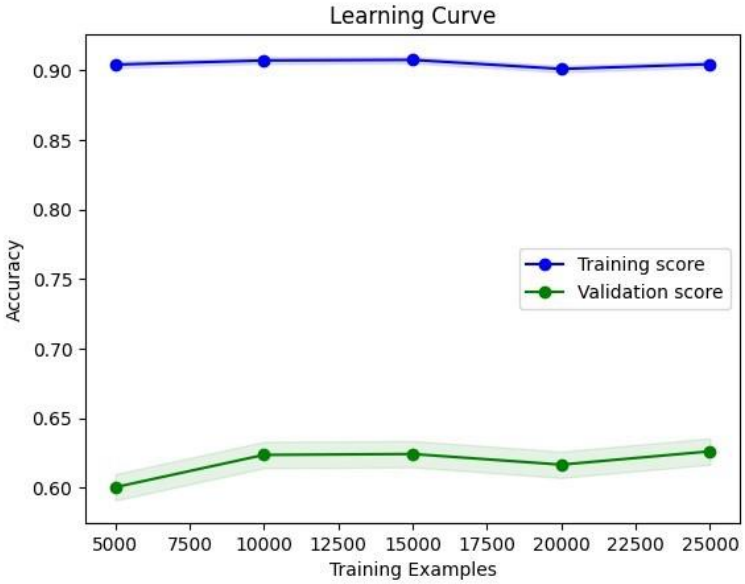
Results, Comparisons and Conclusions

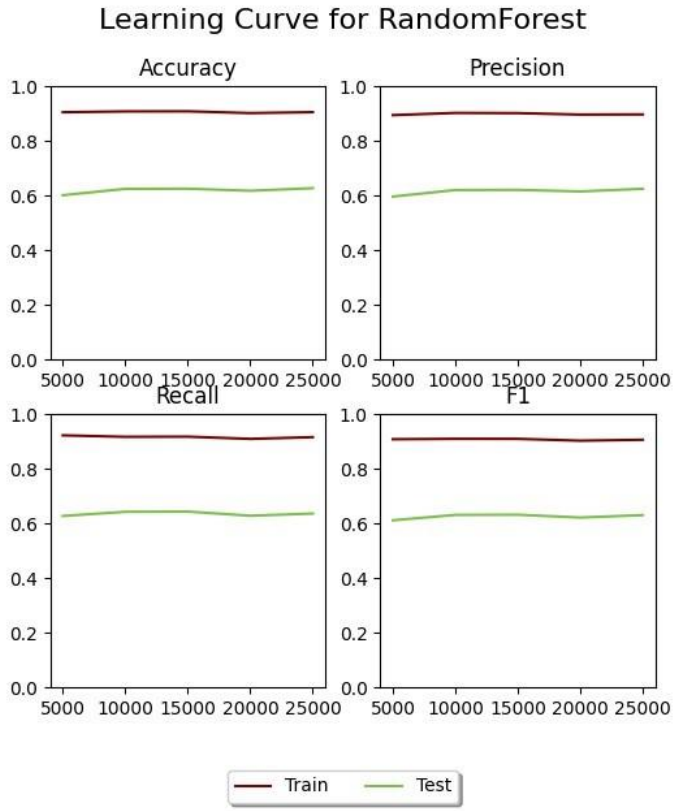
The results we analyze were made from experiments that had the following hyperparameters **m=500, n=50, k=0**. These hyperparameters were found after experimenting with various combinations of values and selecting this combination with the greatest accuracy. As shown in the diagrams with the results of the **Random Forest** algorithm, the accuracy in train data and test data is remarkably close. As we increase the training dataset, the algorithm's performance on test data increases. So, our algorithm does learn and train, since the concept of "experience" is now felt and in fact responds particularly well to evaluation data. By training and evaluating the algorithm of the SkLearn library, and after comparing it with our own algorithm, we noticed (*as shown below in the charts*) that the percentages are quite close. From both the diagrams as well as the arrays (*and/or better the difference table*), the approximations are almost identical.

Πίνακας Διαφοράς για τον RandomForest και τον SKLearn's Random Forest

	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.100000	0.110000	0.130000	0.080000	0.030000	0.180000	0.090000	0.130000
10000	0.140000	0.100000	0.180000	0.060000	0.020000	0.200000	0.110000	0.120000
15000	0.140000	0.120000	0.170000	0.080000	0.060000	0.190000	0.120000	0.130000
20000	0.130000	0.120000	0.180000	0.090000	0.030000	0.210000	0.110000	0.140000
25000	0.140000	0.100000	0.180000	0.080000	0.070000	0.170000	0.130000	0.120000

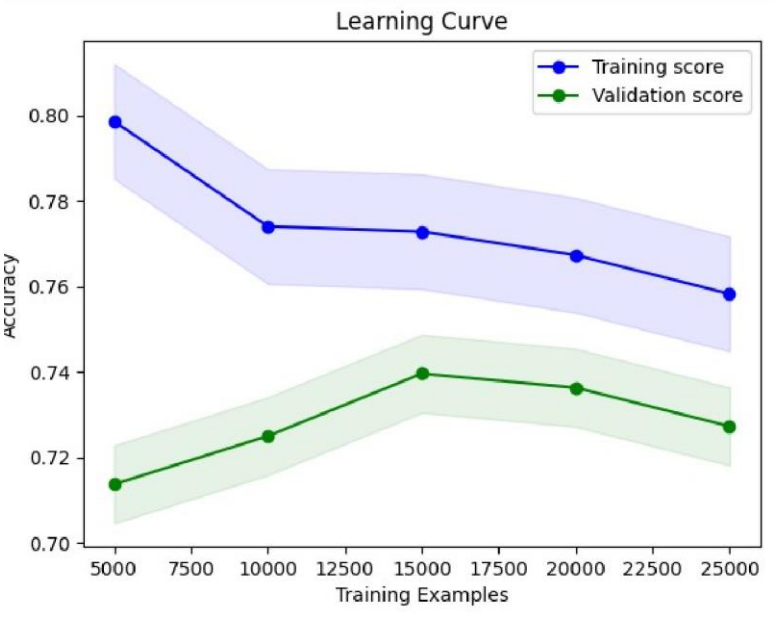
Learning, F1, Precision, Recall Curves for Random Forest

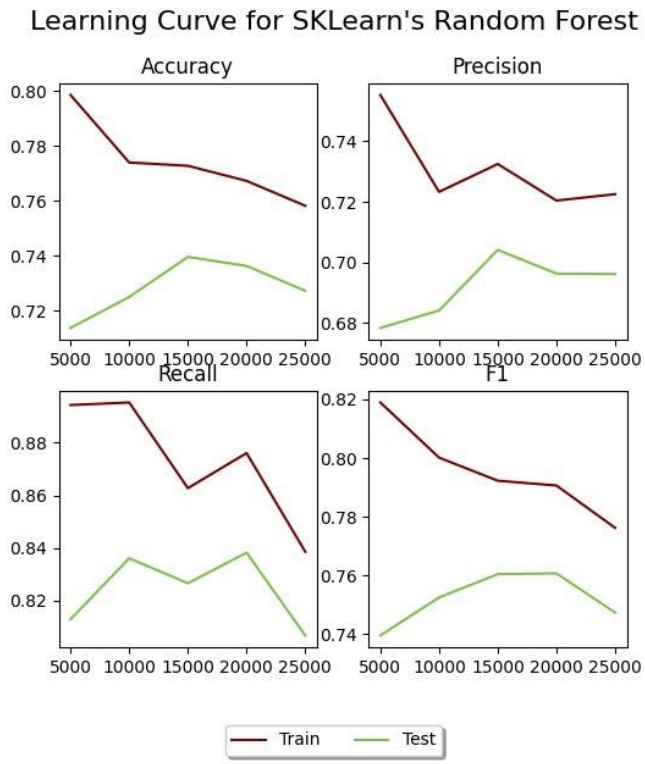


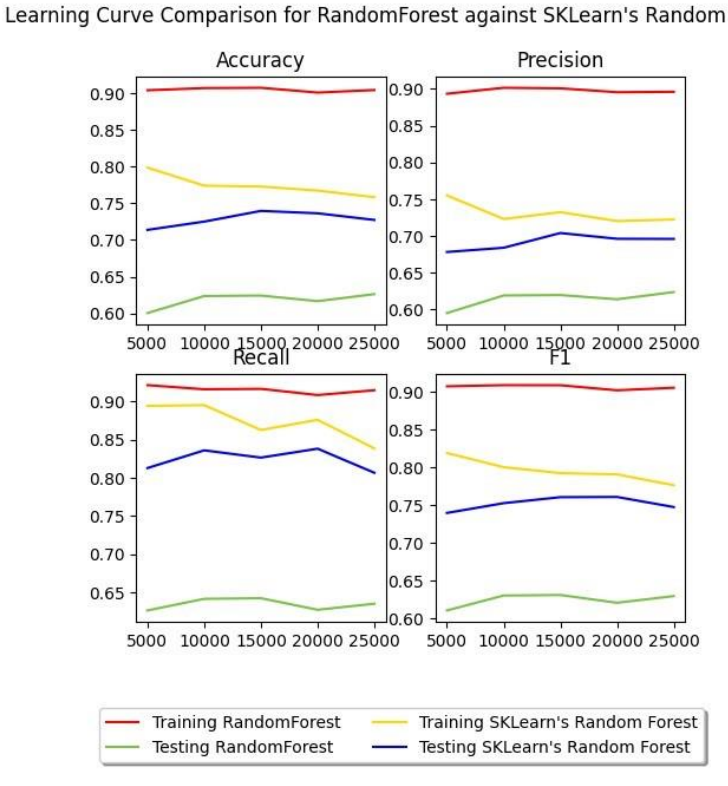


	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.90	0.60	0.89	0.60	0.92	0.63	0.91	0.61
10000	0.91	0.62	0.90	0.62	0.92	0.64	0.91	0.63
15000	0.91	0.62	0.90	0.62	0.92	0.64	0.91	0.63
20000	0.90	0.62	0.90	0.61	0.91	0.63	0.90	0.62
25000	0.90	0.63	0.90	0.62	0.91	0.64	0.91	0.63

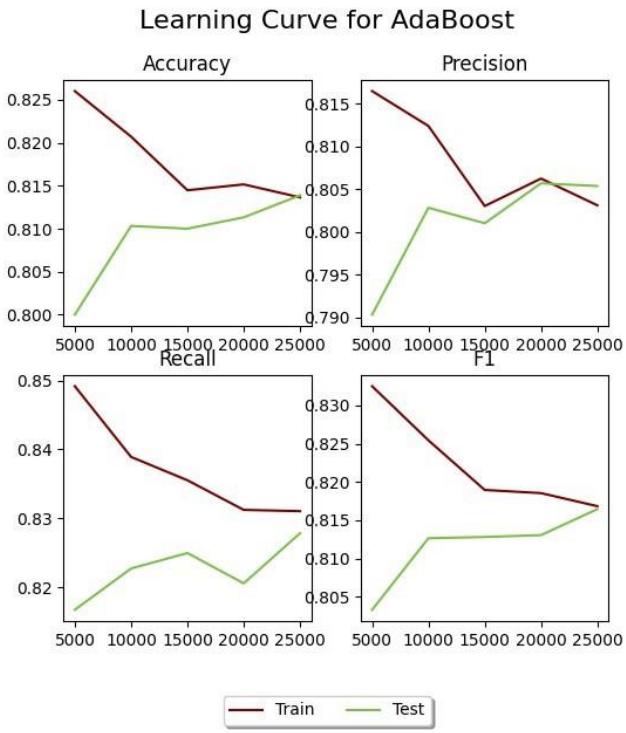
Learning, F1, Precision, Recall Curves for SkLearn's Random Forest

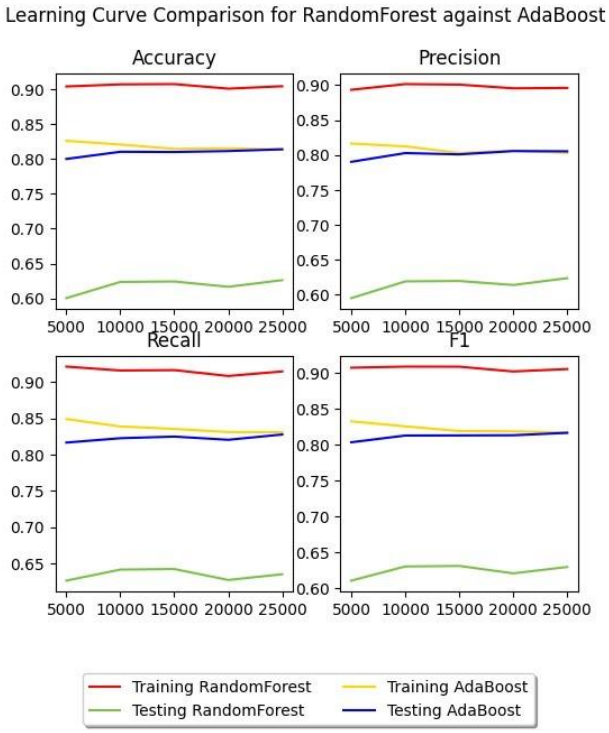






Learning, F1, Precision, Recall Curves for SkLearn's AdaBoost

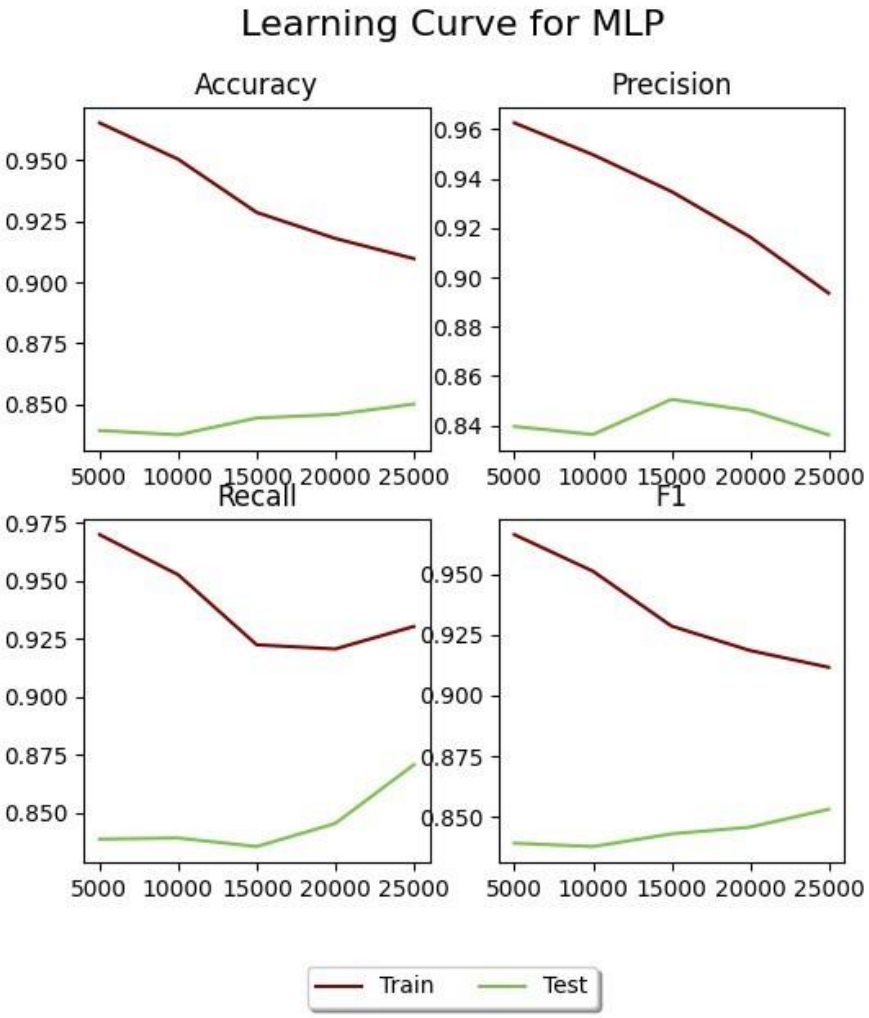




Πίνακας Διαφοράς για τον RandomForest και τον AdaBoost

	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.070000	0.200000	0.070000	0.190000	0.070000	0.190000	0.080000	0.190000
10000	0.090000	0.190000	0.090000	0.180000	0.080000	0.180000	0.080000	0.180000
15000	0.100000	0.190000	0.100000	0.180000	0.080000	0.180000	0.090000	0.180000
20000	0.080000	0.190000	0.090000	0.200000	0.080000	0.190000	0.080000	0.190000
25000	0.090000	0.180000	0.100000	0.190000	0.080000	0.190000	0.090000	0.190000

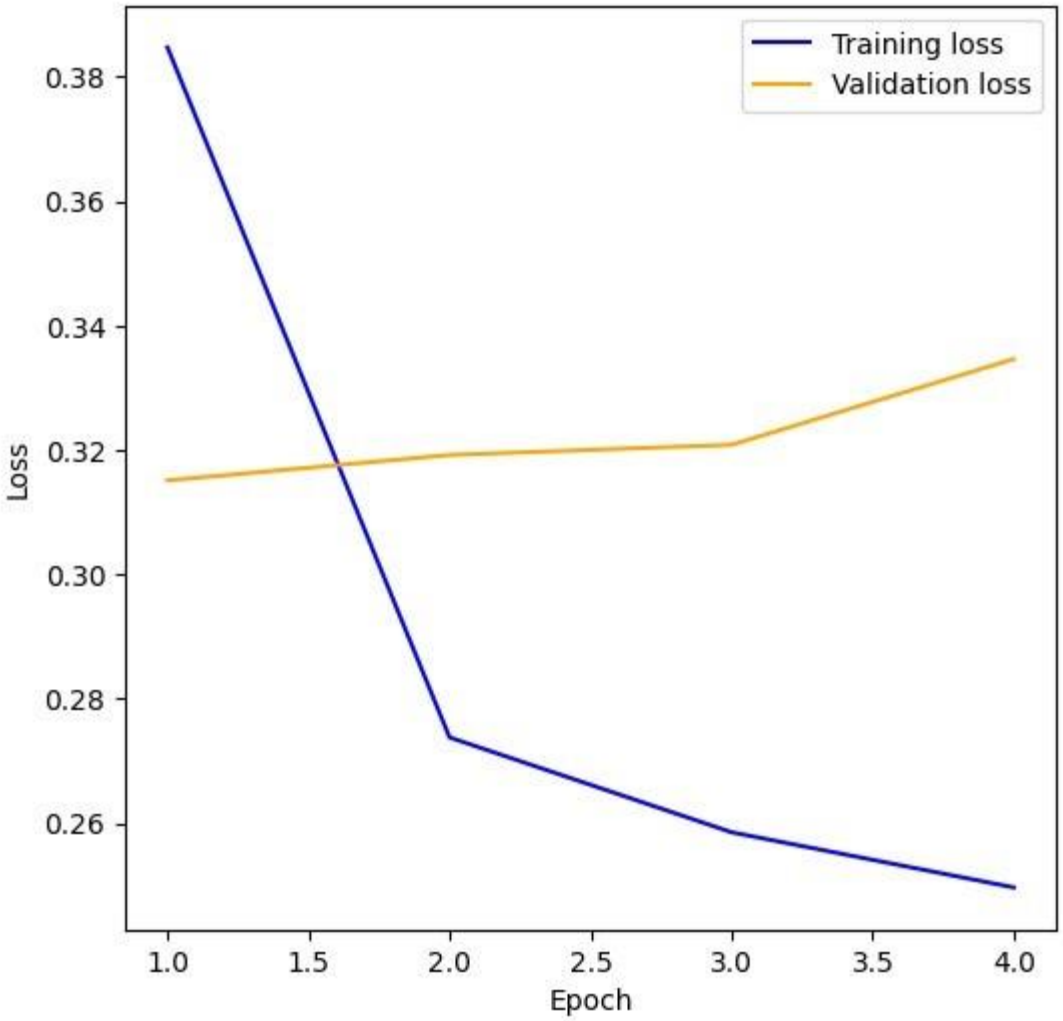
Part C



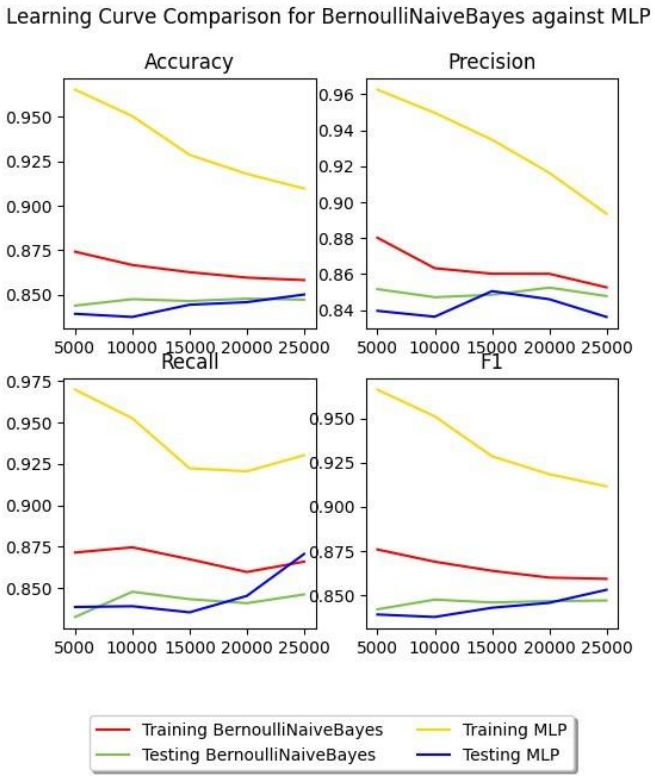
	Train Accuracy	Test Accuracy	Precision Train	Precision Test	Recall Train	Recall Test	F1 Train	F1 Test
5000	0.97	0.84	0.96	0.84	0.97	0.84	0.97	0.84
10000	0.95	0.84	0.95	0.84	0.95	0.84	0.95	0.84
15000	0.93	0.84	0.93	0.85	0.92	0.84	0.93	0.84
20000	0.92	0.85	0.92	0.85	0.92	0.85	0.92	0.85
25000	0.91	0.85	0.89	0.84	0.93	0.87	0.91	0.85

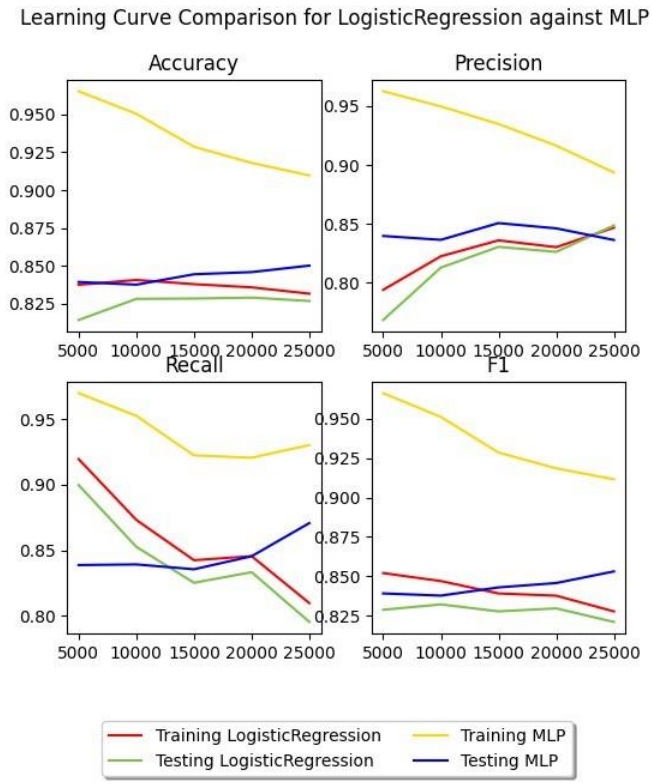
As we observe, the neural network algorithm performs better in all evaluation functions (accuracy, adjusted accuracy, recall, F1). This applies both at the level of our own implementations, as well as at the level of implementations of the SkLearn library. The reason is that the **seasons** help the neural network to **learn better during training** as well as to **perform better during evaluation**.

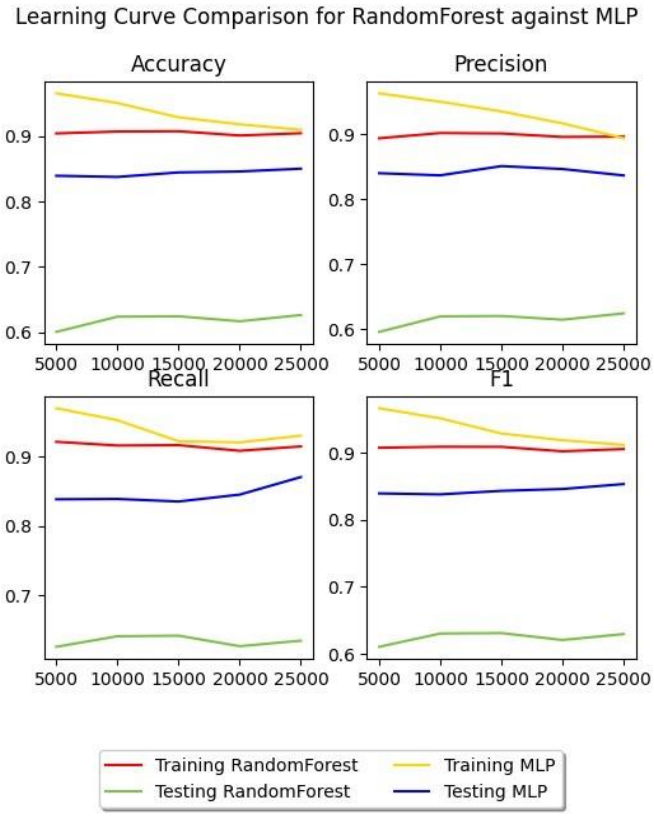
Loss Curve over epochs



Learning, F1, Precision, Recall Curves for SkLearn's algorithms







Epilogue

This project was carried out in the course **"Artificial Intelligence"**, which is a required core course of the 3rd year of the Department of Computer Science. The pdf has been written exclusively by the students of the group, as well as all the code implemented. The course notes, as well as the tutorials, were helpful in this effort. Useful information was also taken from the books by **S.Russell** and **P.Norvig "Artificial Intelligence, a modern approach"** 4th American edition and **I.Vlahava, P.Kefalas, N.Vassiliadis, F.Kok**koras and **H.Skellariou "Artificial Intelligence"** 4th edition.