Assignment 1 in Artificial Intelligence II 2022-2023

Dimitris Orinos ic121006- 7115132100006

In this homework the main purpose is to develop a sentiment classifier using logistic regression for the dataset imdb-reviews.csv which is provided. The work that I have done in order to solve this problem is consisting of:

1st step: Selection of features

Firstly I read the file imdb-reviews.csv through the python library pandas. I decided to use review as features to get better accuracy. What I actually did was to put df[‘review’] to a single string and then preprocess the resulting string with the procedure described in the next section.

2nd step: Preprocessing

In order to get better results in text regression, The steps I took during preprocessing were the following:

• removal of all the numbers as well as punctuation

• conversion of all the words into lower case words. For example I want words such as “Company” and “company” in two different documents, to be translated both as “company” so as to contribute in the union.

• removal of white spaces

• removal of single characters from the start

• substitution of multiple spaces with single space

• removal of prefixed ‘b’

Implementation details:

- For the removal of punctuation and numbers, we imported re module in order to perform regular expression operations such as '[^a-zA-Z]’.
- For the removal of white spaces, we used method strip() for strings
- For the conversion into lower case, we used method lower()
- Removal of stopwords. Some words such as “the” will be appeared many times in the documents without showing something important about the document’s category. We didn’t want these words to be counted in the union, so we removed them with the help of stopwords in nltk.corpus library, through the nltk.download(“stopwords”) and through the parameter of CountVectorizer (explained in step 3) stop_words=stopwords.words(“English”) , where a built-

in stop word list for English is used. There is no problem because all of the words are in English.

2nd step: Creating bag of words

In order to use LogisiticRegression model of scikit learn, first i had to convert texts to vectors. For that case we used the well-known method "Bag of words" and especially CountVectorizer of sklearn.feature_extraction.text. More specificially we use the parameters: max_features=1000, the top max_features ordered by term frequency across the corpus.min_df=0.01 which I use in case of building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words , max_df=0.7 where correspondingly is the case of cut-off i.e when building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. and stop_words=stopwords.words('english') [1].

Also i use TfidfTransformer from sklearn.feature_extraction.text in order to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus, instead of raw frequencies [2]. The total time of data preprocessing (1st and 2nd step): 1 minute and 8 sec.

3rd step : Logistic Regression model. Training the data set using cross-validation

Firstly about the ratings in our dataframe, I give the value 1 for positive reviews and the value 0 (score between 0.0 and 4.0) for negative reviews (score between 7.0 and 10.0). In order to create the Logistic Regression classifier it is used the LogisticRegression() model from sklearn library and more specifically the hyperparameters [3]:

- max_iter: Maximum number of iterations taken for the solvers to converge. In this program max_iter = 500.
- solver: Algorithm to use in the optimization problem. The values that are used: 'lbfgs', 'sag', 'liblinear' and 'saga'.

- penalty: It is related with the regularization terms such as L1 (Manhatan distance) and L2 (Euclidean distance) [4]. The values that are used: 'l1', which is supported only by and 'liblinear' and 'saga' solvers and 'l2' which is supported only by lbfgs', 'sag', 'liblinear' and 'saga' solvers.
- class_weight: Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one. The "balanced" mode, which is used in this solution, uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).
- C : Inverse of regularization strength; must be a positive float. Smaller values specify stronger regularization. Is a float value equal to 0.8 for this model.
- tol: Tolerance for stopping criteria. It is float and in this solution equal to 1e-4, which is the default value for sklearn library.

In order to train our data set the method of cross validation is used, which is a method where each model is evaluated once per validation set after it is trained on the rest of the data. By averaging out all the evaluations of a model, you get a much more accurate measure of its performance [5].

In k-fold cross validation which is used for small data sets, our dataset is split into small disjoint folds subsets called folds. Now we choose one of those k folds as a test set, train our classifier on the remaining k − 1 folds, and then compute the error rate on the test set. Then we repeat with another fold as the test set, again training on the other k−1 folds. This sampling process is done k times and the average validation set score of the k rounds is calculated and should then be a better estimate than a single score [5,6].

In sklearn library there is the K-Folds cross-validator KFold(). The parameters of the validator that are used to make the cross validation are [7]:

- n_splits: Integer value which reperesent number of folds , n_splits = 5 for this exercise.
- shuffle: Boolean value. Whether to shuffle the data before splitting into batches, where shuffle = True.
- random_state: When shuffle is True, affects the ordering of the indices, which controls the randomness of each fold, otherwise this parameter has no effect. It is an random integer value, equal to 220 in this exercise, but generally this parameter could take the value None.

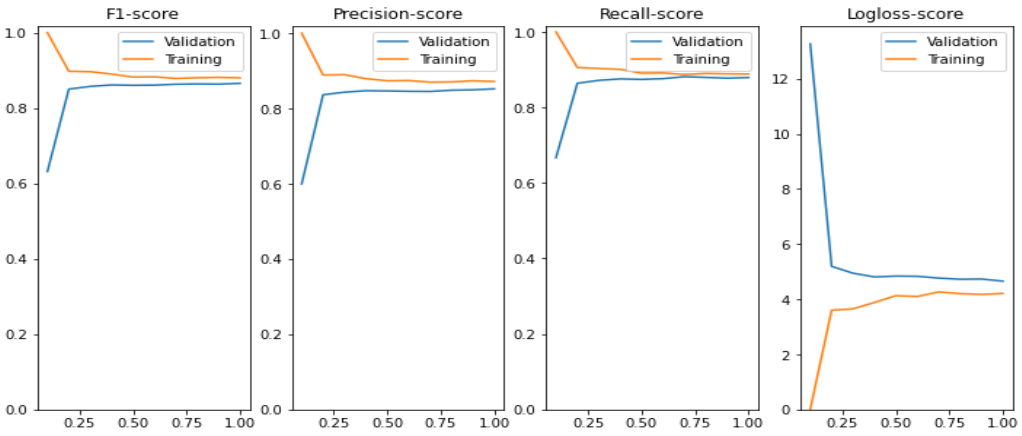
4th step: Final results and explanations

This is the most crucial step in our classification model. Firstly, through KFold that it is mentioned before the model is fitting through fit(X,Y) for each of the five (k = 5) training and validating set occurs from k-fold validation. Later the predictions for each train and validation set for each fold are made through predict() and calcuation for the predicted data of the metrics of precision, i.e a metric which measures the percentage of the items that the system detected, recall i.e a metric which measures the percentage of items actually present in the input that were correctly identified by the system and f1-score, which comes from a weighted harmonic mean of precision and recall. Also such Logistic regression is used it is also calculated the cross entropy function loss function (Log_loss from sklearn library) [4,8].

The next important issue is the learning curves, which is a very good way to prove if our model is overfitting or underfitting. We plot this curves for all the performance metrics that are mentioned before. Below there are results for some combinations of the hyperparameters of logistic regression model and their learning curves. For all these examples only the results of the 1st fold is shown, because all the other folds have very similar results and also are very near to an average estimate of the model performance.

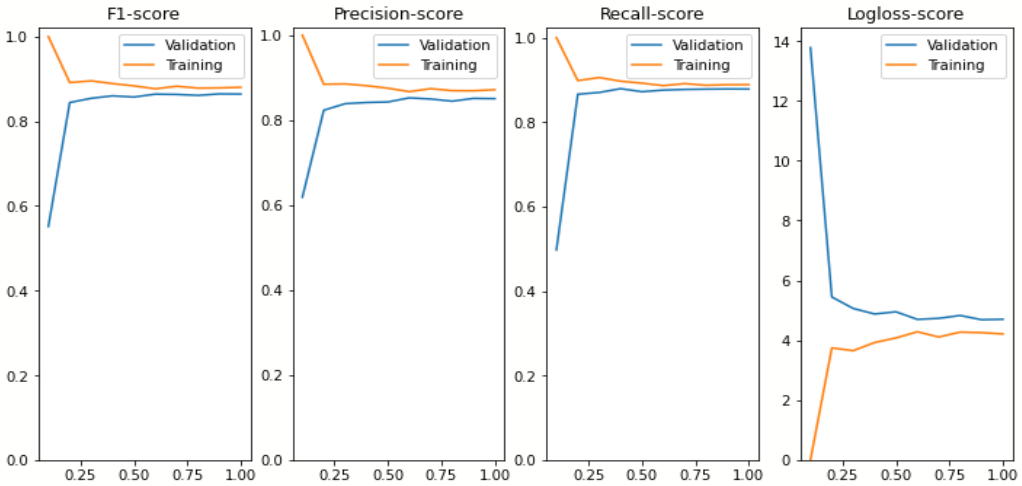
- max_iter = 500, penalty = 'l2', solver = 'liblinear', class_weight = 'balanced', tol = 1e-4, C = 0.8 ,time running: 6 seconds , time plotting of learning curves: 35 seconds.

Performance metric	1st fold value
f1-score(train)	0.8785072411992636
f1-score(validation)	0.8664101154480484
Precision (train)	0.8687428664601337
Precision (validation)	0.8608258684727987
Recall (train)	0.8884936075597554
Recall (validation)	0.8720672864099159
Loss function (train)	4.240900446306007
Loss function (validation)	4.661755461384083



- max_iter = 500, penalty = 'l2',class_weight = 'balanced', tol = 1e-4, solver = 'sag', C = 0.8, time running: 1 minute , time plotting of learning curves: 4 minutes and 29 seconds

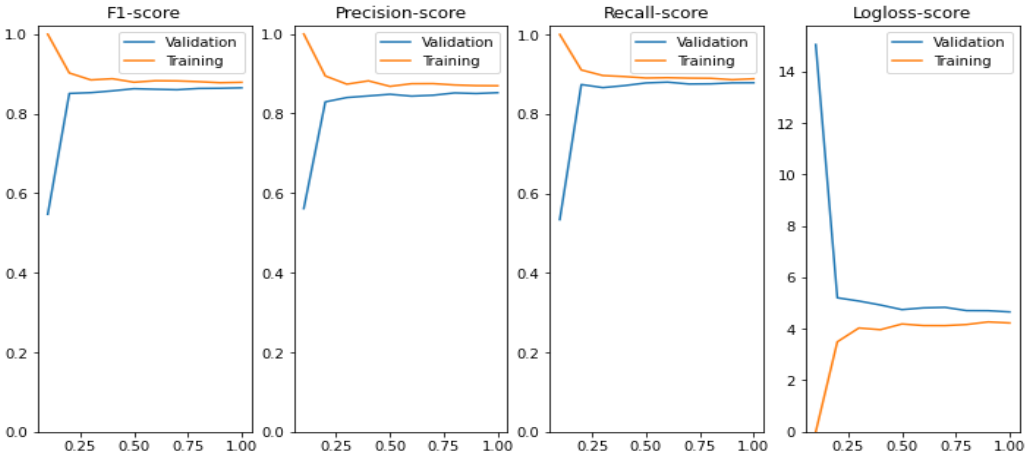
Performance metric	1st fold value
f1-score(train)	0.8785072411992636
f1-score(validation)	0.8664101154480484
Precision (train)	0.8687428664601337
Precision (validation)	0.8608258684727987
Recall (train)	0.8884936075597554
Recall (validation)	0.8720672864099159
Loss function (train)	4.240900446306007
Loss function (validation)	4.661755461384083



- max_iter = 500, penalty = 'l2', class_weight = 'balanced', tol = 1e-4, solver = 'lbfgs' (default value), class_weight = 'balanced', C = 0.8 , time running: 19 seconds , time plotting of learning curves: 1 minute and 28 seconds

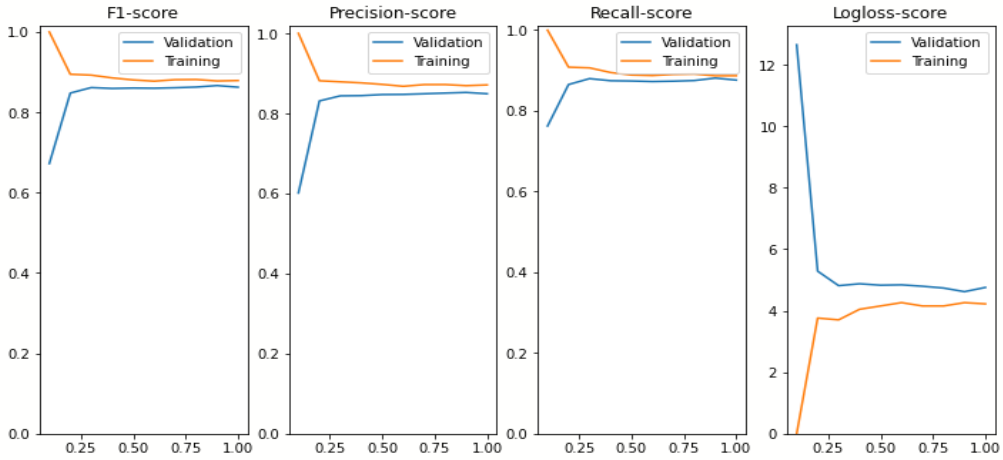
Performance metric	1st fold value
f1-score(train)	0.8785313839727382
f1-score(validation)	0.8664101154480484
Precision (train)	0.8687900858788998
Precision (validation)	0.8608258684727987
Recall (train)	0.8884936075597554

Recall (validation)	0.8720672864099159
Loss function (train)	4.239941173518351
Loss function (validation)	4.661755461384083



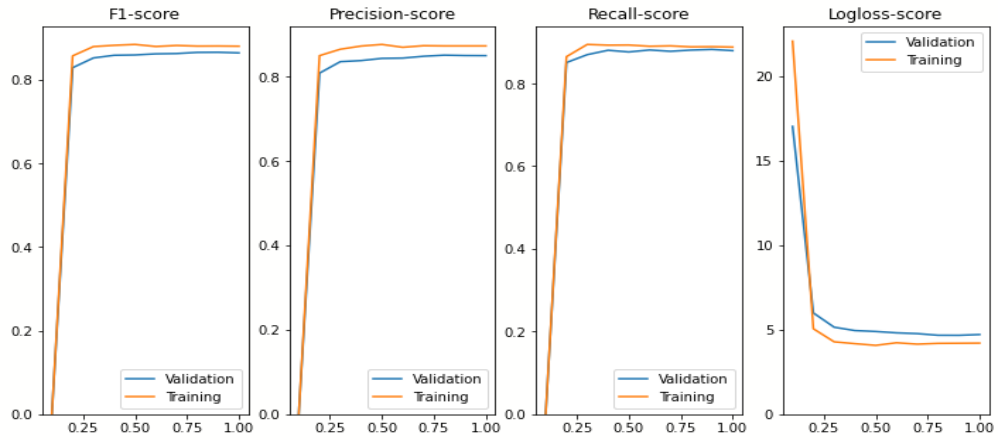
- max_iter = 500, penalty = 'l2', class_weight = 'balanced', tol = 1e-4, C = 0.8 , solver = 'saga', class_weight = 'balanced', time running: 54 seconds , time plotting of learning curves: 3 minutes and 55 seconds

Performance metric	**1st fold value**
f1-score(train)	0.8784830997526794
f1-score(validation)	0.8664101154480484
Precision (train)	0.8686956521739131
Precision (validation)	0.8608258684727987
Recall (train)	0.8884936075597554
Recall (validation)	0.8720672864099159
Loss function (train)	4.241859719093663
Loss function (validation)	4.661755461384083



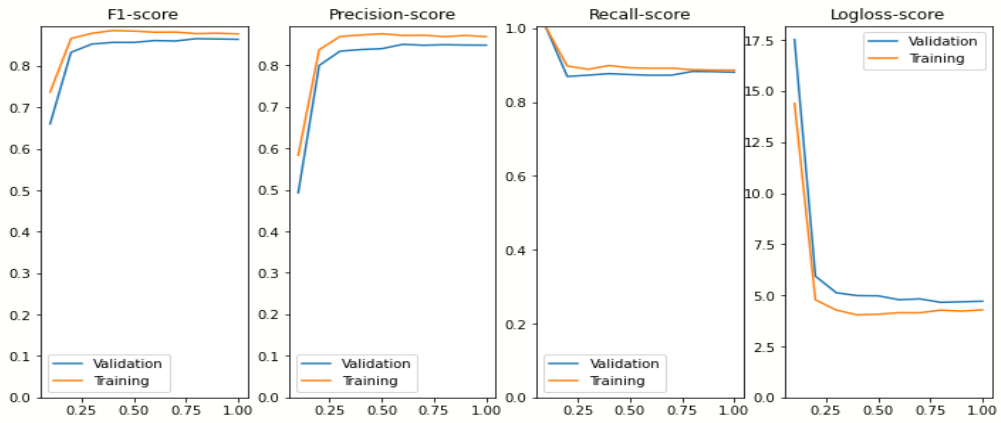
- max_iter = 500, penalty = 'l1', class_weight = 'balanced', tol = 1e-4, solver = 'liblinear', C = 0.8, class_weight = 'balanced', time running: 5 seconds , time plotting of learning curves: 34 seconds.

Performance metric	1st fold value
f1-score(train)	0.8783505154639174
f1-score(validation)	0.8642246106602327
Precision (train)	0.8689148762578188
Precision (validation)	0.8565217391304348
Recall (train)	0.8879933296275708
Recall (validation)	0.8720672864099159
Loss function (train)	4.244737337590584
Loss function (validation)	4.750003655987932



- max_iter = 500, penalty = 'l1', class_weight = 'balanced', tol = 1e-4, solver = 'saga', C = 0.8, class_weight = 'balanced', time running: 1 minute and 22 seconds , time plotting of learning curves: 6 minutes and 20 seconds

Performance metric	1st fold value
f1-score(train)	0.8784121835225555
f1-score(validation)	0.8643194033124931
Precision (train)	0.8689291347147441
Precision (validation)	0.8567079799956512
Recall (train)	0.8881045025013896
Recall (validation)	0.8720672864099159
Loss function (train)	4.242818836429948
Loss function (validation)	4.746166777961678



From all the above results it is easy to understand that cross validation and especially kfold cross validation method makes this model to probably generalize much more better on other similar csv files, than splitting the test set randomly only once [5,8]. Firstly from the results of the 1st fold and its learning curves, it is observable that the model does not overfitting and underfitting, which means that the model performs well on the training data and on other validation data [8]. The metrics of precision, F1-score and recall in all examples give very good values between 85% and 88%.

Also in all the learning curves in these three metrics have always bigger train values than validation values ,which occurs from our training test, something that is expected, regardless of speaking for decrease or increase of values. When L2 regularization or L1 regularization term is used which penalize the weights with high values, although that could fit our data perfectly, in L2 it is a more quadratic variation of the ratings, where in second case it is a more linear validation, but in both cases the loss function is decreased and overfitting probabilities are shrinked[3,4,6].

About the cross entropy loss function, in all cases in 1st and in all folds the validation set has bigger loss than the remaining training set each time, something that is expected because from the definition of this loss function the model's estimation closer to the correct value (the bigger the probability for ratings to 1 or to 0 in our case) the smaller the loss it is, in other case our model has a bigger loss [4,8]. This ascertainment is getting stronger from recall, precision and f1-score metrics.

The solver 'sag', referring to the stochastic gradient descent algorithm which is an online algorithm (processing its input example by example) that minimizes the loss function by computing its gradient after each training example and nudging to the opposite direction of the gradient such as all the gradient descent algorithms [4,8]. Something that helps a lot the performance of this algorithm to get higher is the shuffling of the training set not picking randomly instances, before spiliting the dataset of the exercise's csv file into batches through cross validation ,although it converges more slowly [3,4].

Although it is a fast algorithm for large datasets, the number of samples is a little bigger than the number of features and so the algorithm in this case, it is not extremely fast The solver 'saga' the solver of choice for sparse multinomial logistic regression. where multinomial logistic regression label each observation with a class k from a set of K classes, under the stipulation that only one of these classes is the correct one [4,8].

The ‘liblinear’ solver uses a coordinate descent (CD) algorithm, which cannot learn a true multinomial (multiclass) model; instead, the optimization problem is decomposed in a “one-vs-rest” fashion so separate binary classifiers are trained for all classes. Generally it is not so fast for larger datasets compared to ‘sag’ and ‘saga’ solvers, but due to the fact that samples are not so much bigger than features, ‘sag’ and ‘saga’ solvers have lost some of their fastness. Unfortunately it penalizes the intercept, but such as “lbfgs” solver, it has more robustness in unscaled datasets. This robustness is not a characteristic of sag and saga solvers, because in other cases the performance metrics will be more decreased [3].

The “lbfgs” is an optimization algorithm that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm, which belongs to quasi-Newton methods[3]. The “lbfgs” solver is recommended for use for small data-sets but for larger datasets its performance suffers [9,10].

5th step: Predicting the test set

After training the model and preprocessing of the test set as the steps 1,2 and 3 explains, the next step is to predict the test set. Validation set is set as none through the variable test_df, in order to evaluate each test (validation) set, only by passing the path of whichever test set, for example test_df = pd.read_csv(…..), into the variable test_df. The reviews are stored and merged in a big list and afterwards, transform of CountVectorizer is used into that list in order to create Xval, because fit_transform is used in order to train our model. Also Tfidf Transformer is used (again transform) and finally precision, f1-score and recall are predicted through the classifier ,which is created in the 3rd step, as we described in the 4th step. About the ratings the same process is followed as it is described in the 3rd step.

References

1. sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.1.3 documentation
2. sklearn.feature_extraction.text.TfidfTransformer — scikit-learn 1.1.3 documentation
3. 1.1. Linear Models — scikit-learn 1.1.3 documentation
4. 5.pdf (stanford.edu)
5. 4.pdf (stanford.edu)
6. Introductory Concepts of Machine Learning (uoa.gr)
7. sklearn.model_selection.KFold — scikit-learn 1.1.3 documentation
8. Regression (uoa.gr)
9. https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm
10. http://www.fuzihao.org/blog/2016/01/16/Comparison-of-Gradient-Descent-Stochastic-Gradient-Descent-and-L-BFGS/