Text Analytics: 2nd Assignment

“Text Classification with Linear Classifiers”

Authors: **Andreadis Georgios**, f3352101 **|** **Gatos Rafail**, f3352103
Koumentakos Agis, f3352108 **|** **Moscholios Filippos-Michail**, f3352116
Stavrou Andreas, f3352120

Abstract

The present document constitutes the report for the 2nd assignment “Text Classification with Linear Classifiers” in Text Analytics. The code, on which the present report is based, can be found in the following link: Colab. The analysis starts by loading the chosen data as well as preprocessing it to reach the required form for the analysis. Afterwards the dataset is split to train, dev and test sets. Using the train and dev sets we implement grid search in order to optimize the parameters of numerous classifiers used (Dummy, Naive Bayes, Logistic, SVM and KNN). Additionally, we optimize the dimensionality of our dataset as well as using SVD to reduce the features in the case of the SVM model. Further on the metrics (Precision, Recall, F1 and AUC) of all models for the three datasets (train, dev and test) are calculated and presented for both classes (positive/negative). Also, the Precision-Recall Curves show the relation between precision and recall during testing. Finally, the Learning Curves of the models are presented showcasing the different behaviors during training, dev and testing.

Data

The dataset used in the assignment is the “IMDB Review Dataset”, found in Kaggle. It includes 50.000 reviews, which belong to either one of two classes: “Positive (1)” or “Negative (0)”. The classes in the training dataset are equiprobable with 25,000 reviews per class.

A splitting schema of training (70% - 35,000 Reviews), development (15% - 7,500 reviews) and test set (15% - 7,500 reviews) was applied. The ratio of Positive to Negative reviews was preserved while applying the splitting schema. To do this we used the *train_test_split* method from the *sklearn* library with *test_size* = 0.15 the first time in order to split the original dataset in 85% and

15%. The second time we split the train set (85%) with the same method with *test_size* = 0.17647, because 0.85*0.17647 ≈ 0.15. So we have the final split we wanted.

Data Manipulation

Before applying any classification algorithm, and before the splitting described above, the dataset was processed by the method ***preprocess***. This method converts the reviews to lowercase, removes special characters, and lemmatizes, if needed, each word. The lemmatization was performed with the *WordNetLemmatizer()* method from the *NLTK* library.

As for the vectorization, we performed feature extraction using the **TF-IDF** vectorizer using the *TfidfVectorizer* method provided by the *sklearn* library. This function also performs **feature selection** with the *max_features* attribute. We selected 15,000 features. Also, we used to cut-off some very frequent words of the English language with the *stop_words* attribute, and we used only unigrams and bigrams from the development of the vocabulary. Theoretically, the **TF-IDF** vectorizer takes as input a text and calculates for each word the product of two probabilities. The first is the TF (Term frequency) term which is the number of occurrences of the word in the text divided by the total number of the tokens of the text (usually we take the logarithm of this value). The second is the IDF (Inverse Document Frequency) term, which is the logarithm of the number of documents in corpus divided by the number of corpus documents containing the word of interest (shows how rare or frequent is the word in the language generally). We perform this vectorization in the ***grid_search_method***, which is used for the tuning of the parameters. This method is analyzed in detail below.

Classifiers and Hyperparameters Tuning

We have created a ***grid_search_classify*** method which does most of the data validation and fitting in an automated way to save some time and experiment with more parameters with ease. This method takes as inputs the classifier, the parameters to be tuned, the train reviews and the train labels and performs grid search, with the predefined development set. The grid search uses a pipeline with the TF-IDF and the classifier. The parameters to be tuned are different for each classifier, but for the TF-IDF we search between 5000, 10000 and 15000 features.
For the optimization we used sklearn's *GridSearchCV* function along with the development set we previously created. In order to achieve this and avoid having a different validation set for the grid search, we used sklearn's *PredefinedSplit* method, found in here.
Also, for the classifiers that we choose to optimize we report below the best parameters after tuning.

For the training of the models we used the following classifiers:

Dummy Classifier

It was used as a reference point which is validated by the equiprobability of the classes of the training dataset. The strategy of this classifier is to classify each point (text with review in our case) to the most frequent class. Because the classes are totally balanced, it can classify each new review to either class, and it chooses the negative class. To implement this classifier we used the function *DummyClassifier()* from the sklearn library.

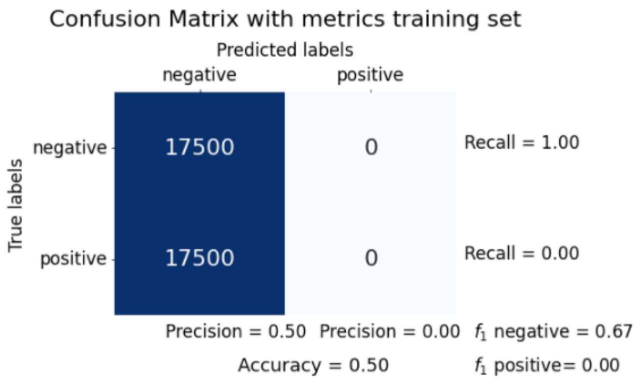


Figure 1 Confusion Matrix for test set Dummy Classifier

Multinomial Naive Bayes:

Calculates the probabilities

$$P(C = 0|X) = \frac{P(C=0) \cdot \prod_{i=1}^{m} P(X_i = x_i|C=0)}{P(X)}$$

and

$$P(C = 1|X) = \frac{P(C=1) \cdot \prod_{i=1}^{m} P(X_i = x_i|C=1)}{P(X)}$$

and assigns the new point (tf-idf vector resulting from the text) to the class with the higher probabilities of the two. It is trained with the TF-IDF vectors using the function *MultinomialNB()* from the sklearn library. The F1-Scores were (apx.) 0.89, 0.87 and 0.87 for the train, development and test data respectively. In addition, we had about 6 times more True Positives and negatives than the False ones.

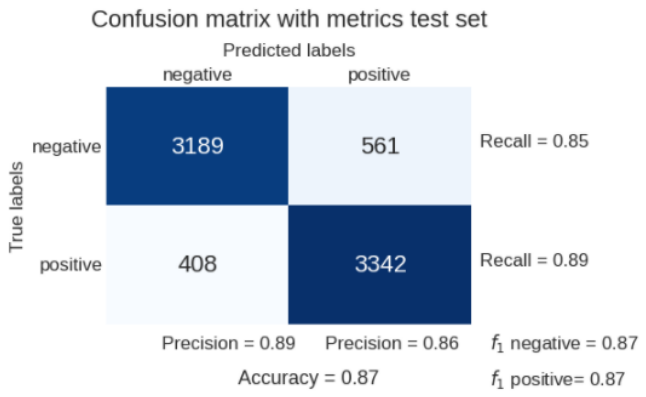


Figure 2Confudion Matrix for test set Naive Bayes

Logistic Regression:

The logistic regression uses the sigmoid function. The probability of the new observation belonging to the positive class is $P(C_+|x) = \frac{1}{1+e^{-w \cdot x}}$ and the probability of belonging to the negative class is $P(C_-|x) = 1 - P(C_+|x)$. To estimate the weights *w* we use the loss function

$$l(w) = \sum_{i=1}^{m} y^{(i)} log(\, P(C_+|x^{(i)}; w)) + \\ + (1 - y^{(i)}) log(P(C_-|x^{(i)}; w)\,)$$

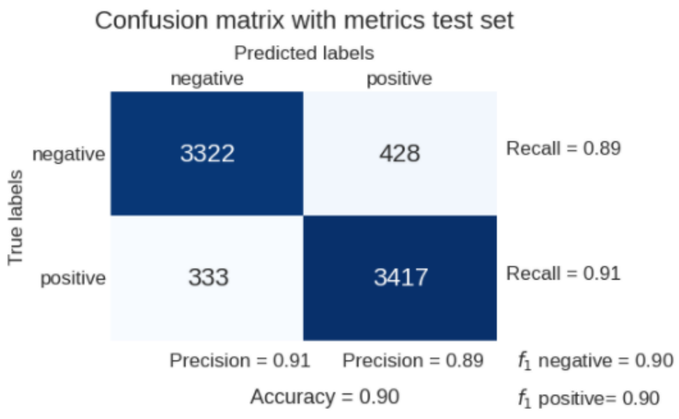


Figure 3 Caption. Confusion Matrix for test set Logistic Regression

To implement this method we used the function *LogisticRegression()* from the sklearn library.
It was trained with the TF-IDF vectors. The F1-Scores were 0.93, 0.89 and 0.90 for the train, development and test data respectively.

SVM:

With this classifier we try to find the "maximum-margin hyperplane" that divides the two classes which is defined so that the distance between the hyperplane and the nearest point of each class is maximized. For non linear separable problems, soft margin and/or non linear kernels can be used.
We trained this classifier with and without dimensionality reduction using the TF-IDF vectors.

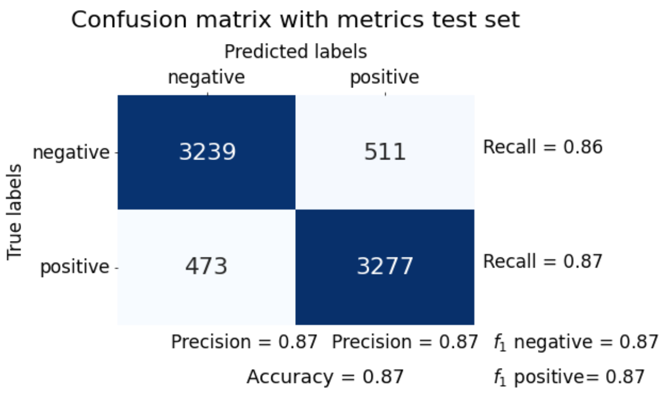


Figure 4Confusion Matrix SVM with Dimensionality Reduction

The dimensionality reduction was achieved through the Truncated SVD method using 50 components. SVD is a linear algebra method which uses the singular values of a matrix in order to create a matrix of smaller dimension which maintains as much information as we want about the initial matrix. Specifically, the SVD of an (m,n) matrix A is a factorization of the form $A = U\Sigma V^T$ where U is a (m,m) unitary matrix, Σ is a (m,n) rectangular diagonal matrix with non-negative number on the diagonal (singular values), and V is a (n,n) unitary matrix.

KNN-Classifier:

This classifier, for any new (unlabeled) data point x finds the classes of the k closest data points of the training set and assigns x to the “majority class”.
We used k = 5 for a first attempt which yielded F1-Scores (apx.) 0.88 and 0.81 on the training and development dataset respectively.

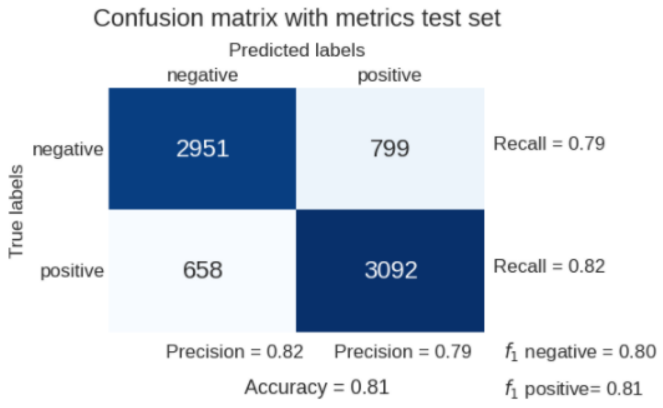
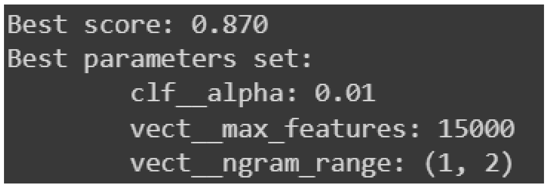


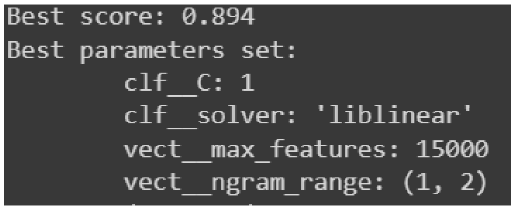
Figure 5 Confusion Matrix kNN with k = 5

Optimizing Naive bayes and Logistic Regression Classifiers



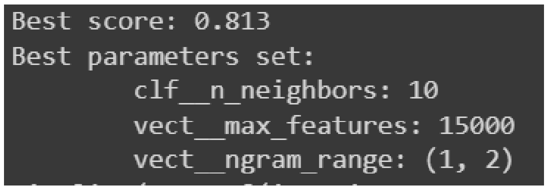
```
Best score: 0.870
Best parameters set:
        clf__alpha: 0.01
        vect__max_features: 15000
        vect__ngram_range: (1, 2)
```

Figure 7 Naive Bayes Best Score and Parameters set



```
Best score: 0.894
Best parameters set:
        clf__C: 1
        clf__solver: 'liblinear'
        vect__max_features: 15000
        vect__ngram_range: (1, 2)
```

Figure 6 Logistic Regression Best Score and Parameters set



```
Best score: 0.813
Best parameters set:
        clf__n_neighbors: 10
        vect__max_features: 15000
        vect__ngram_range: (1, 2)
```

Figure 8 kNN Best Score and Parameters set

Optimizing SVM Classifier

The nature of this classifier made it computationally demanding to optimize its parameters through grid search. Therefore, the tuning was done manually through trial and error and we used SVD to simplify the problem.
We tried different feature dimensions utilizing the ‘Truncated SVD’ function and concluded that above 50 dimension features the performance (f1 score) on the dev subset does not improve significantly, whereas the computational time required does.
As far as the SVM-kernel is concerned the ‘rbf’ proved superior to ‘linear’ and ‘poly’, achieving better f1-scores than the first and significantly shorter computational time than the second.

Based on the above, our manual tuning led to an SVM classifier with an ‘rbf’ kernel and a 50-dimension dataset, that scores approximately 0.87 f1-score.

SVD and Dimensionality reduction with respect to f1

The above procedure of feature dimensionality reduction for SVM to run smoothly, lead us to the idea or searching the minimum feature dimensionality of the vectorized dataset, without sacrificing in performance.

For this to be correctly depicted, we created a graph showcasing the macro averaged f1 score of the model for different feature dimensions and compared it to the optimum model performance that was found previously through grid search. The respective graph is found on the right.

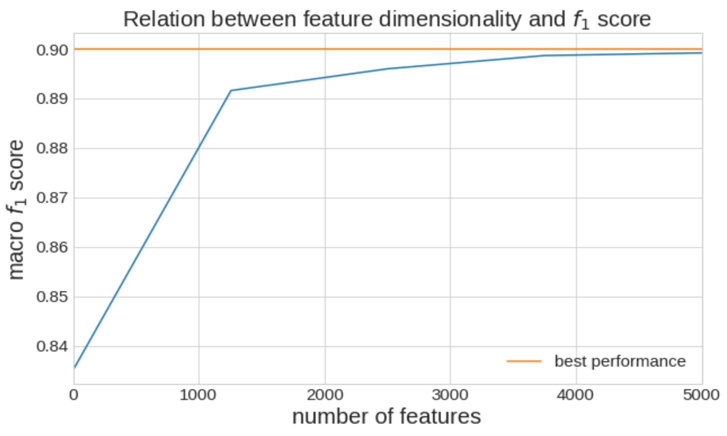


Figure 9 Deciding on the optimum dimensionality based on Macro f1 score

From the graph it can be observed that feature dimensionality above 1000 does not result in significant performance improvement of the model (macro f1 score) as it is already to 0.89 almost equal to the optimal 0.90.

Diagnosing

Learning Curves - Metrics and Final evaluation on which model to use.

Precision-Recall curves:

At this part we construct the Precision-Re
below we present the corresponding curves
the AUC score for each classifier on testing

Learning curves:

Afterwards, Learning curves showcasing m
develop, for training, developing and testin
for our models: Dummy classifier,

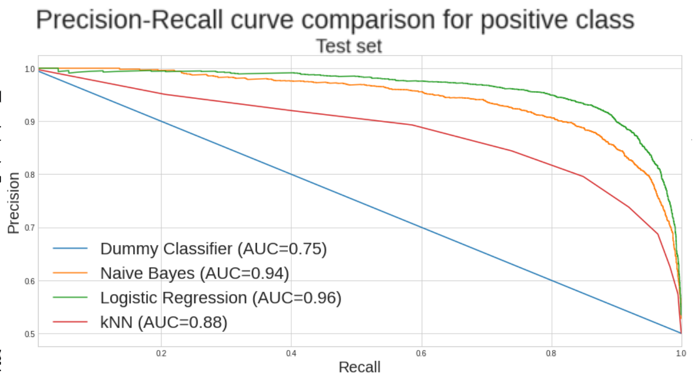


Figure 10 Precision Recall AUC curves (all models) positive class

Logistic Regression, Naive Bayes and Knn. Also, present the learning curves for training and developing subsets, for SVM with dimensionality reduction, we didn't add the curve for testing subset because it's time consuming.

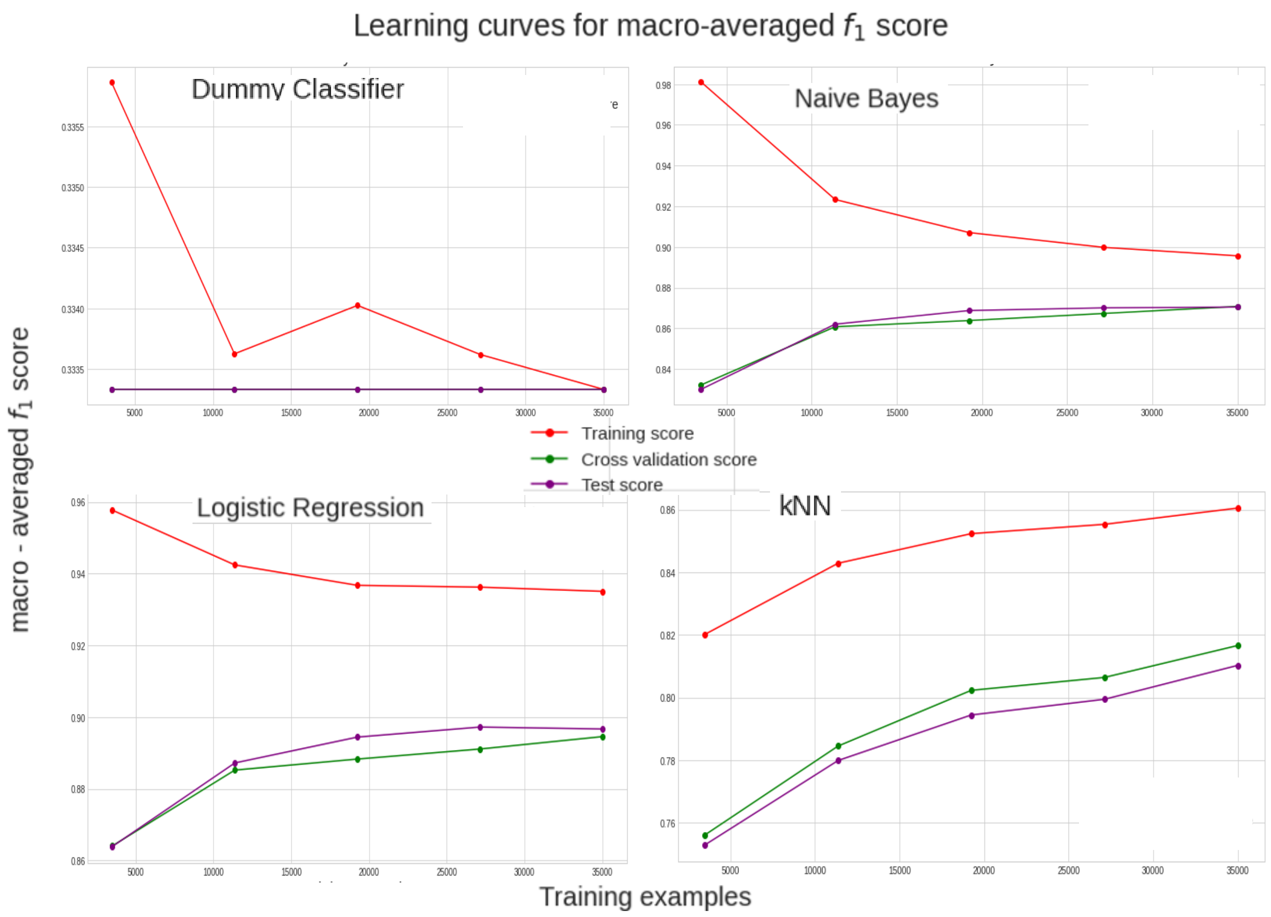


Figure 11Caption. Learning Curves on Training, Developing and Testing subsets

Results

Before using the unseen dataset we applied the same preprocess as with the training data.
We present the precision, recall, f1, auc (of precision-recall curve) in a table for each of the classifiers for each of the datasets (train, dev, test).

Training Subset

	Negative				**Positive**				**Macro-Averaged metrics**			
	P	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**
Dummy	0.5	1.00	0.67	0.75	0.00	0.00	0.00	0.75	0.25	0.50	0.33	0.75
NB	0.90	0.88	0.89	0.96	0.88	0.91	0.89	0.96	0.89	0.89	0.89	0.96
LR	0.94	0.93	0.93	0.98	0.93	0.94	0.93	0.98	0.93	0.93	0.93	0.98
KNN	0.88	0.84	0.86	0.94	0.85	0.89	0.87	0.94	0.86	0.86	0.86	0.94

Development Subset

	Negative				**Positive**				**Macro-Averaged metrics**			
	P	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**
Dummy	0.5	1.00	0.67	0.75	0.00	0.00	0.00	0.75	0.25	0.50	0.33	0.75
NB	0.90	0.87	0.88	0.96	0.88	0.90	0.89	0.95	0.89	0.89	0.89	0.95
LR	0.93	0.92	0.93	0.96	0.92	0.93	0.93	0.98	0.93	0.93	0.93	0.97
KNN	0.88	0.84	0.86	0.90	0.85	0.88	0.87	0.94	0.86	0.86	0.86	0.92

Test Subset

	Negative				**Positive**				**Macro - Averaged metrics**			
	P	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**	**P**	**R**	**F1**	**AUC**
Dummy	0.5	1.00	0.67	0.75	0.00	0.00	0.00	0.75	0.25	0.50	0.33	0.75
NB	0.89	0.85	0.87	0.95	0.86	0.89	0.87	0.94	0.87	0.87	0.87	0.94
LR	0.91	0.89	0.90	0.96	0.89	0.91	0.90	0.96	**0.90**	**0.90**	**0.90**	**0.96**
KNN	0.84	0.78	0.81	0.90	0.80	0.85	0.81	0.88	0.82	0.82	0.82	0.89

Conclusion

It is worth mentioning that all the models of the present research performed quite good in correctly classifying the data. However, there was significant difference as far as the computational efficiency is concerned. Specifically, the kNN classifier was very inefficient as it uses brute force in order to classify, which in our case where the dataset is big, kNN is impractical. As far as the SVM is concerned the dimensionality of the dataset played a significant role, with "high dimensionality" features making it impossible to "run". This led us in using SVD in order to reduce the feature dimensionality and making SVM less computationally demanding. Moreover, we used SVD I order to find the optimum feature/performance ratio, concluding that a dataset of 1000 feature dimensionality performs almost optimally as far as the macro averaged f1 score is concerned and significantly less computationally demanding.
Furthermore, the present research dealt with the "learning" behavior of all models. In detail, we observe that Naïve baye ang logistic Regression Classifiers behave normally, meaning that it becomes less possible for the to overfit as the size of the dataset increases. On the other hand the kNN classifier shows signs of overfitting, as its performance in the training set increases when the size of the dataset is increased whereas the difference between the performance in the train vs the dev and test set remains the same , not converging as it "should". This could be explained by the fact that the kNN with a fixed k parameter equal to 10 becomes gradually very "small" compared to the size of the dataset, leading to overfit. This could of course be prevented by increasing the k-parameter, however practically this will lead to a very computationally demanding classifier.

Contributor	Theoretical Understanding	Code Implementation	Report
Georgios Andreadis	Main Con.	Main Con.	Main Con.
Rafail Gatos	Review Con.	Review Con.	Sec.Con.
Agis Koumentakos	Main Con.	Main Con.	Main Con.
Filippos Moscholios	Sec.Con.	Main Con.	Main Con.
Andreas Stavrou	Main Con.	Main Con.	Sec.Con.

Main Con.: Main contributor, played a fundamental role in the respective task
Sec. Con.: Secondary contributor, played an important role in parts of the respective task
Review Con.: Review contributor, having observed and given feedback in the respective task

For the proper functioning of the team, we agreed that each contributor should be main contributor in at least one of the tasks.