

# Report

## Executive summary.

In Task 1, successfully tackled data preprocessing, feature engineering, and model training using TF-IDF vectorization and a Logistic Regression. Through detailed experimentation and grid search with cross-validation optimal hyperparameters are identified, ensuring model generalization while maximizing accuracy. The evaluation on the validation set, utilizing classification metrics and confusion matrix insights, confirms the model's strong classification abilities across distinct categories. The inclusion of entity features enriches prediction relevance. Overall, the model effectively identifies topics with nuanced performance evaluation, meeting client expectations for accuracy and generalization. A scalar performance metric like F1-score would provide informative insights into overall algorithm performance.

In Task 2, I employed a multi-step approach involving feature engineering and machine learning algorithms. Initially, a pipeline with TfidfVectorizer and an MLP classifier was utilized, yielding promising results. Subsequently, optimization through GridSearch identified an SGDClassifier with an alpha of 0.01 and L1 penalty as the best configuration, achieving a high accuracy of 95% on the test set. Incorporating PCA for dimensionality reduction and class weights to handle imbalance further refined the model, maintaining high accuracy while improving recall for unclear texts. This comprehensive strategy effectively met the client's criteria for successfully identifying text clarity, balancing precision and recall across classifications.

## 1. Data exploration and assessment and Cleaning

### 1.1 Missing Values

Initially, exploring the given dataset is one of the significant parts of data preprocessing part. In this task cleaning part is processed before splitting data. Considering this it would be much easier to preprocess the data on-the-go. To detect the missing values around all the features, 'data.isna().sum()' function used.

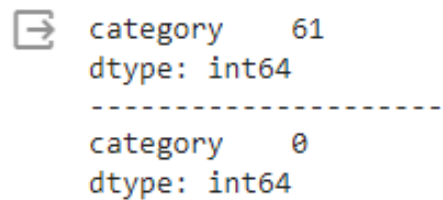
```
par_id      0
paragraph   0
has_entity   0
lexicon_count  0
difficult_words  18
last_editor_gender  0
category     61
text_clarity 9338
dtype: int64
```

*Figure 1*

Exploring the dataset, comprising 9357 entries, reveals that the 'difficult\_word' column has 18 missing values. Deciding to drop these rows, as they contain a small portion of the dataset and have minimal impact on the algorithm. This approach ensures data integrity and does not significantly affect the model's performance. Utilizing 'data\_dropna()' facilitates this process seamlessly. Please also see Figure 1.

The 'text\_clarity' column is for the second task therefore this can be same until second task.

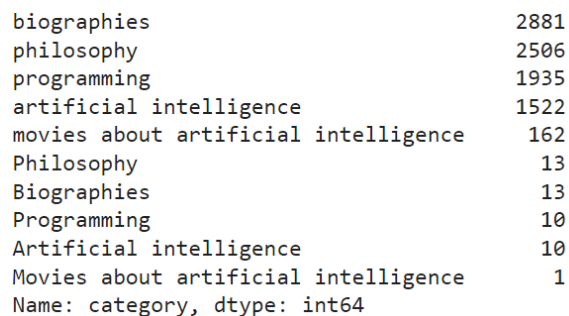
## 1.2 Simple Imputer for 'category' column



```
category    61
dtype: int64
-----
category     0
dtype: int64
```

**Figure 2**

The 'category' feature contains 61 missing values, critical for the task where it serves as the target column. Dropping these rows might not be efficient. Instead of this, employing SimpleImputer with the 'most\_frequent' strategy is deemed more advanced. This strategy imputes the most frequent category type to handle missing values effectively. By using SimpleImputer, data integrity is preserved, ensuring that the 'category' feature remains strong for following tasks.



biographies	2881
philosophy	2506
programming	1935
artificial intelligence	1522
movies about artificial intelligence	162
Philosophy	13
Biographies	13
Programming	10
Artificial intelligence	10
Movies about artificial intelligence	1
Name: category, dtype: int64	

**Figure 3**

The 'category' column is target column for the first task, in that case distribution of this column might give some insights. In Figure 3, the distribution between the paragraph topics can be seen. When data is collected, variations in capitalization can lead to the same categories being treated as different ones due to case sensitivity. For instance, 'Biographies' and 'biographies' would be seen as two separate categories.

```

biographies                2894
philosophy                 2519
programming               1945
artificial intelligence    1532
movies about artificial intelligence  163
Name: category, dtype: int64

```

**Figure 4**

In Figure 4, converting all text to lowercase standardizes the data, ensuring uniform treatment of categories regardless of their original format. The resulting distribution offers insight into the dataset's frequent topics, guiding following analysis and decision-making processes.

```

ORG_YES_PRODUCT_NO_PERSON_YES_  2987
ORG_NO_PRODUCT_NO_PERSON_NO_    2788
ORG_YES_PRODUCT_NO_PERSON_NO_   1445
ORG_NO_PRODUCT_NO_PERSON_YES_   1348
ORG_YES_PRODUCT_YES_PERSON_YES_   292
ORG_YES_PRODUCT_YES_PERSON_NO_    124
ORG_NO_PRODUCT_YES_PERSON_YES_    64
ORG_NO_PRODUCT_YES_PERSON_NO_    42
data missing                      24
Name: has_entity, dtype: int64
-----
ORGYESPRODUCTNOPERSONYES        2987
ORGNOPRODUCTNOPERSONNO          2788
ORGYESPRODUCTNOPERSONNO         1445
ORGNOPRODUCTNOPERSONYES         1348
ORGYESPRODUCTYESPERSONYES        292
ORGYESPRODUCTYESPERSONNO         124
ORGNOPRODUCTYESPERSONYES         64
ORGNOPRODUCTYESPERSONNO          42
Name: has_entity, dtype: int64

```

**Figure 5**

In Figure 5, the 'has\_entity' column's data and distribution are reviewed. Noticeably, missing values are represented as 'data missing' strings. Rows containing this value are dropped to ensure analysis on complete data. The column's values are simplified by removing underscores for readability and usability in programming contexts. For instance, 'ORG\_YES\_PRODUCT\_NO\_PERSON\_YES\_' becomes 'ORGYESPRODUCTNOPERSONYES'. This enhances visual clarity and reduces potential issues with algorithms. The modified distribution of 'has\_entity' values illustrate the cleaned data structure.

### 1.3 Taking only necessary columns in 'task2\_data' (Task2)

The 'task2\_data' dataset is derived from the 'temp\_dataset' by removing the 'paragraph' and 'par\_id' columns, which are unnecessary for machine learning model. The 'paragraph' column has already been cleaned and assigned to 'clean\_lemmatized\_text'. After these modifications, the 'head()' method is employed to display the initial rows of the updated dataset, confirming the changes.

## **1.4 Setting Conditions to take 100 datapoints for prototype (Task2)**

In the second task, a hundred data points are randomly selected and stored in a dataframe called 'task2\_sample', averaging with the project specifications. Before splitting the dataset, mean values for 'difficult\_words' and 'lexicon\_count' are calculated serving as threshold for determining text clarity. This classification introduces a new column, 'clarity', assuming texts with more difficult words and a higher lexicon count are less clear to readers. This approach aids in distinguishing between 'clear\_enough' and 'not\_clear\_enough' texts, a crucial step for the task's objectives.

## **2. Data Splitting**

In the preprocessing and cleaning purposes of data preparation for tasks one and two, all necessary steps are typically performed before splitting the data into training, validation, and test sets to prevent data leakage and to be sure that the model evaluation is as accurate as possible. Data leakage happens when information from outside the training dataset is used to create the model, leading to confuse the model's performance on unseen data.

All preprocessing and cleaning steps, excluding the TF-IDF vectorizing, were completed before to data splitting. These steps include handling missing values, duplicates, distribution of categories, labelling and encoding the necessary data. The reason for this sequence is to make sure the training, validation and test sets all undergo the same transformations and to prevent any problem that could affect the model's performance.

### **2.1 First Task**

For the first task, after completing the general preprocessing and cleaning, the dataset 'task1\_data' was split into training, validation and test sets using an 80-20 split for the train-test splitting, then taking 10% of the original dataset size for validation from the training set. The 'stratify' parameter ensures that the category distribution remains consistent across all splits, which is critical for maintaining model integrity and for the evaluation of the model's performance to be more accurate.

### **2.2 Second Task**

In the second task, the dataset is divided into features 'X' and the target variable 'y', where 'X' contains all columns except for 'clarity' column which is the target column for this task. This splitting is necessary for supervised learning, as the goal is to predict the target variable based on the features. The dataset has been split into training and test sets using a standard 80-20% split. Then training set is divided into a new training set and a validation set. The use of a validation set allows us to compromise the test set, which should only be used for final model evaluation.

### 3. Data encoding

#### 3.1 One-Hot Encoding for 'has-entity' column

One-hot encoding is a preferred method for converting categorical variables into a numerical format, essential for machine learning models. It creates new columns to represent each unique category, assigning binary values (1 or 0) to indicate the presence or absence of a category. The 'ColumnTransformer' function along with 'OneHotEncoder' from scikit-learn, is utilized specifically for the 'has\_entity' column, preserving other columns in the dataset. This transformation ensures straightforward feature names and replaces the original column with new ones, facilitating accurate interpretation by machine learning algorithms. It prevents misinterpretation of categorical variables as having numeric relations, thereby enhancing the model's performance.

#### 3.2 Label Encoder for 'category' column

The encoding process is vital to convert categorical columns into a numerical format for machine learning algorithms. Utilizing the 'LabelEncoder' tool from scikit-learn's preprocessing module accomplishes this task by assigning unique integer labels to each category. For example, categories like 'artificial intelligence' and 'biographies' are mapped to integer values 0, 1, 2, 3, and 4. Applying the 'fit\_transform' method to the 'category' column generates numerical labels, ensuring compatibility with machine learning algorithms and enhancing model performance.

The final print statement displays this mapping, which in the output example given is:

{0: 'artificial intelligence', 1: 'biographies', 2: 'movies about artificial intelligence', 3: 'philosophy', 4: 'programming'}. This dictionary is particularly useful for understanding predictions or model coefficients in terms of the original categorical values.

#### 3.3 Label Encoder for 'clarity' column (Task2)

As 'clarity' column has been created for second task, there was a need to encode this column on-the-go. The 'LabelEncoder' object identified as 'lblEncoder' is fitted to the 'clarity' column of the 'task2\_sample' dataset. The 'fit' method learned the mapping between textual labels and numerical labels. By executing these steps, 'clarity' column of 'task2\_sample' dataset has been effectively encoded into a numerical format, preventing the significant data in a form that is compatible with machine learning models.

### 4. Task 1: topic classification

#### 4a. Model building.

The logistic regression model is chosen for topic classification due to its simplicity, efficiency with high-dimensional data like TF-IDF vectors, and interpretability. It allows for regularization strength (parameter C) and penalty type (L1 or L2 norm), balancing model complexity and generalization performance.

Hyperparameter Selection:

Hyperparameter	Description	Chosen Value
C	Inverse of regularization strength	10
Penalty	Regularization penalty (L1 or L2 norm)	L2
Max_iter	Maximum number of iterations for optimization	500

Due to these parameters, the best score was *0.92126*.

Detailed experimentation, including Grid Search and Cross-Validation, optimized the logistic regression model. Grid Search identified optimal hyperparameters, such as 'Best Parameters: {'C': 10, 'max\_iter': 500, 'penalty': 'l2'}', balancing fitting and regularization. The resulting 'best\_model' variable reflects these optimal parameters. Through cross-validation, hyperparameters were fine-tuned for optimal performance. The model effectively prioritizes fitting without overfitting, leveraging logistic regression's probability modeling for precise classification. Results in the Model Evaluation section confirm the approach's efficacy, demonstrating accurate topic classification while maintaining strength.

#### 4b. Model evaluation.

After tuning hyperparameters to find the best settings, the model's performance is assessed on a validation set. Following the identification of the best model parameters, the aim was to interpret the model's predictions in a meaningful way. To perform this, a dictionary named 'category\_names' maps numerical labels back to their original category names. This mapping is to convert numeric labels in the validation set 'y\_val' and the predicted labels 'y\_pred\_val\_grid' into readable category names.

The 'best\_model' (extracted from the grid search's 'best\_estimator') is used to make predictions on the validation set transformed by TF-IDF vectorization 'X\_val\_tfidf'. The accuracy and detailed performance metrics of these predictions are evaluated using 'accuracy\_score' and 'classification\_report' from scikit-learn.

Accuracy on Validation Set (Best Model): 0.9108910891089109

Classification Report on Validation Set (Best Model):

	precision	recall	f1-score	support
0	0.88	0.82	0.85	153
1	0.91	0.94	0.92	295
2	1.00	0.88	0.93	16
3	0.92	0.92	0.92	251
4	0.92	0.93	0.93	194
accuracy			0.91	909
macro avg	0.93	0.90	0.91	909
weighted avg	0.91	0.91	0.91	909

Figure 6

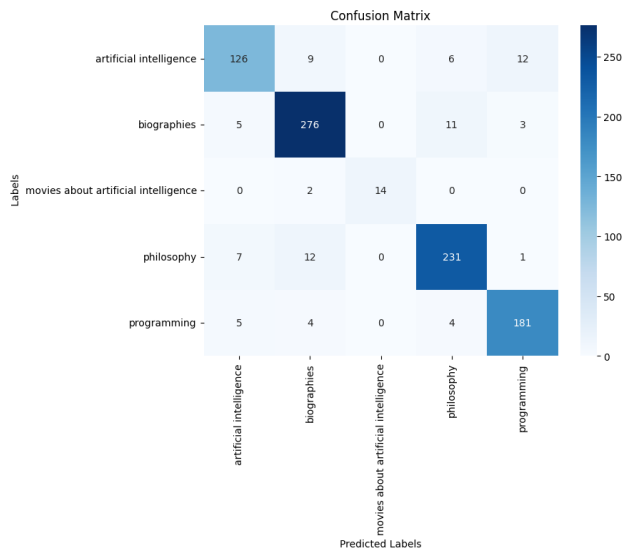


Figure 7

The confusion matrix (Figure 7) from the model's performance on the validation set provides insightful data into the algorithm's classification abilities across five distinct categories. The matrix indicates a strong true positive rate for 'artificial intelligence', 'philosophy', and 'programming' categories, with respective counts of 126, 231, and 181 correct classification.

4c. Task 1 Conclusions

For ongoing performance tracking, the F1 score is recommended as it balances precision and recall, providing a comprehensive view of the algorithm's performance, especially when class

imbalance is present.

As a bullet point, the model is successful according to the client's definition of success. It ensures that no more than 10% of the paragraphs get misclassified into an unrelated class, with only one error ("artificial intelligence" text misclassified as "programming") which the client is willing to overlook.

To further track the algorithm's performance, I would highly recommend the client to use precision-recall curve. This metric provides insight into the exchange between precision and recall across different classification thresholds, allowing the client to arrange the model's behaviour based on their specific requirements and preferences regarding misclassification tolerance.

## 5. Task 2: text clarity classification prototype.

### 5a. Ethical discussion.

There are several ethical concerns and risks associated with using an algorithm to automatically reject user edits based on predicted text clarity. The choice of labels, such as "clear\_enough" and "not\_clear\_enough", may accidentally maintain biases if not carefully selected or if the labelled subset is not representative.

Data hazard labels:

1. Bias and Fairness: Indicates the presence of biases in the dataset that may lead to unfair outcomes, such as favoring certain language styles or penalizing specific demographics.
2. Misinterpretation: Different users may perceive clarity differently based on their backgrounds, knowledge, and preferences.
3. Privacy Concerns: Automated text clarity assessment systems must ensure data privacy and confidentiality to protect users' rights and prevent unauthorized access to or misuse of personal information.

### 5b. Data labelling.

I have labelled the data automatically to create a data frame ('task2\_sample') which includes 100 sample data points. Firstly, mean values for 'difficult\_words' and 'lexicon\_count' are computed across the entire 'task2\_data' dataset. Then, each sample in 'task2\_sample' is assessed against these mean values. By calculating these values, the algorithm was able to perform conditions on the dataset. If a sample exhibits a higher-than-average number of difficult words and lexicon count, it is labelled as 'not\_clear\_enough'. If the values are less than mean value it returns 'clear\_enough'.

```
0    64
1    36
Name: clarity, dtype: int64
```

Figure 7



## 5c. Model building and evaluation.

The solution for the second task (the text\_clarity classification) involves using a pipeline consisting of a preprocessor and a classifier. For the first try (Please see the related part on the notebook submitted.), the preprocessor performs 'ColumnTransformer' with 'TfidfVectorizer' to transform the text data into numerical values and the hyperparameters defined by performing GridSearch. Then MLP classifier implemented from scikit-learn.

Hyperparameters Selection for the first try:

Parameter	Value
'ngram_range'	(1,1)
'max_features'	Default
'alpha'	0.0001, 0.001
'hidden_layer_sizes'	(20,20)

Hyperparameters Selection for second try:

Parameter	Values Tested	Best Value
Alpha	[0.0001, 0.001, 0.01]	0.01
Penalty	[ 'l1', 'l2' ]	L1

The table displays the hyperparameters tuned for the SGDClassifier model using GridSearch. For the 'alpha' parameter, values of 0.0001, 0.001, and 0.01 were tested. The 'penalty' parameter was tested with both L1 and L2 norms. The best-performing configuration was found to be an alpha value of 0.01 with an L1 penalty, achieving a best score of 0.929.

After implementing best hyperparameters the classification reports:

Classification Report for Validation Set:					Classification Report for Test Set:				
	precision	recall	f1-score	support		precision	recall	f1-score	support
0	1.00	0.83	0.91	6	0	0.93	1.00	0.96	13
1	0.80	1.00	0.89	4	1	1.00	0.86	0.92	7
accuracy			0.90	10	accuracy			0.95	20
macro avg	0.90	0.92	0.90	10	macro avg	0.96	0.93	0.94	20
weighted avg	0.92	0.90	0.90	10	weighted avg	0.95	0.95	0.95	20

Table 1 (Classification Reports)

The classification reports depict the model's performance on validation and test sets. The test set shows an impressive 95% accuracy, with high precision, recall, and F1-score for both classes. In the validation set, accuracy remains at 90%, with differences in class performance. Class 0 exhibits higher precision, while class 1 demonstrates better recall. This suggests challenges in classifying unclear text despite overall strong performance. The chosen approach, combining feature engineering and classification algorithms, effectively meets the text classification task's

requirements.

5d. Principal Component Analysis (PCA)

The second model approach incorporates Principal Component Analysis (PCA) and Standardization techniques. PCA is utilized to reduce the dimensionality of the dataset while retaining variance, with the number of components determined by cumulative explained variance. Various models are trained using SGDClassifier across different numbers of principal components, and the model exhibiting the highest validation accuracy is selected. This optimized model achieves a commendable test accuracy of 95%, indicating strong performance in perceptive text clarity. Noticeably, in the validation set class 1 exhibits lower precision and recall indicating potential difficulties in identifying unclear text. (Michael Greenacre, 2022)

The best number of principal components to use is: 9

	precision	recall	f1-score	support
0	0.93	1.00	0.96	13
1	1.00	0.86	0.92	7
accuracy			0.95	20
macro avg	0.96	0.93	0.94	20
weighted avg	0.95	0.95	0.95	20

Figure 8

Provide By combining PCA with standard scaling, the dimensionality of the data is effectively reduced while preserving its essential information, resulting in a more efficient and effective model.

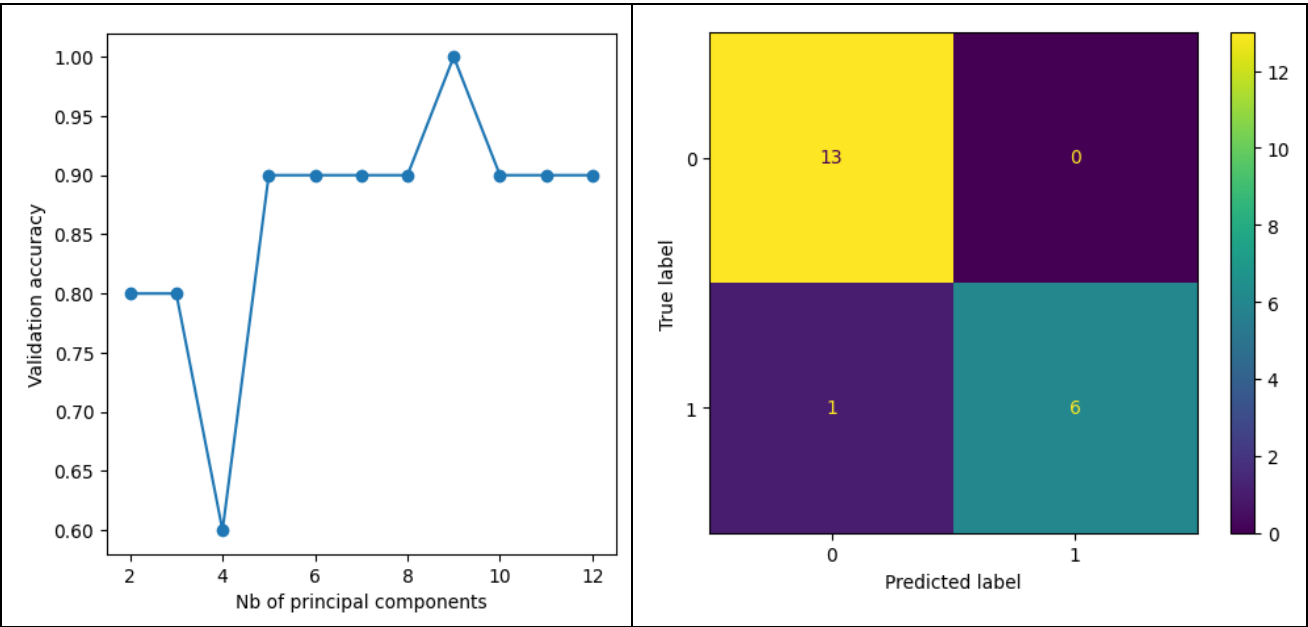


Table 2 (PCA Evaluation Plots)

## 5e. Class Weight

Class weights, computed via 'compute\_class\_weight', address imbalance by assigning higher weights to minority class instances. SGDClassifier, trained with these weights, balances both classes in the loss function. The reports show improved recall for class 1 in validation and test sets. However, precision slightly decreases in validation. Overall, class weights enhance the model's performance. There is a notable improvement in the recall of class 1 in both validation and test sets. However, this improvement comes at a slight cost to precision, particularly in the validation set. (sklearn, n.d.)

Classification Report for Validation Set with Class Weights:				
	precision	recall	f1-score	support
0	0.86	1.00	0.92	6
1	1.00	0.75	0.86	4
accuracy			0.90	10
macro avg	0.93	0.88	0.89	10
weighted avg	0.91	0.90	0.90	10

Classification Report for Test Set with Class Weights:				
	precision	recall	f1-score	support
0	0.93	1.00	0.96	13
1	1.00	0.86	0.92	7
accuracy			0.95	20
macro avg	0.96	0.93	0.94	20
weighted avg	0.95	0.95	0.95	20

**Figure 9 (Class Weights)**

## 5d. Task 2 Conclusions.

The model achieves the client's success criteria by outperforming the majority class baseline. To track performance comprehensively, I recommend using the F1-score as an additional metric due to its balance between precision and recall. For improvement, I suggest a detailed analysis misclassification, focusing on unclear text instances to uncover patterns leading to misclassification. This could involve examining instances where the model incorrectly labels text clarity and identifying patterns or common characteristics that may be leading to misclassification.

## 6. Self-reflection

The "Hyperparameter optimization" section could be enhanced by including advice on advanced optimization like Bayesian optimization. To improve, I would explain the benefits of using Bayesian methods over grid search, such as efficiency and effectiveness in finding optimal parameters.

The "Model Evaluation" section for both first and second tasks could be improved by emphasizing the importance of Stratified Cross-Validation and Additional Evaluation Metrics. Specifically, elaborating on how Stratified Cross-Validation preserves class distribution in each fold would ensure more accurate performance assessments. Introducing metrics like MCC and AUC would provide a more understanding of model performance, especially in scenarios with imbalanced

classes.

## 7. References

Anon., n.d. *sklearn.utils.class\_weight.compute\_class\_weight*. [Online]

Available at: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html)

[learn.org/stable/modules/generated/sklearn.utils.class\\_weight.compute\\_class\\_weight.html](https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html)

[Accessed 03 2024].

Michael Greenacre, P. J. G., 2022. Principal component analysis. *nature*.

Overflow, S., 2023. *How to ignore errors in code and continue working, but print error codes?*.

[Online]

Available at: <https://stackoverflow.com/questions/77351084/how-to-ignore-errors-in-code-and-continue-working-but-print-error-codes>

sklearn, n.d. *sklearn.utils.class\_weight.compute\_class\_weight*. [Online]

Available at: [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html)

[learn.org/stable/modules/generated/sklearn.utils.class\\_weight.compute\\_class\\_weight.html](https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html)

[Accessed 03 2024].