Machine Learning for NLP 1

Exercise 1

Language Identification with Scikit-Learn and Skorch

University of Zurich

Contents

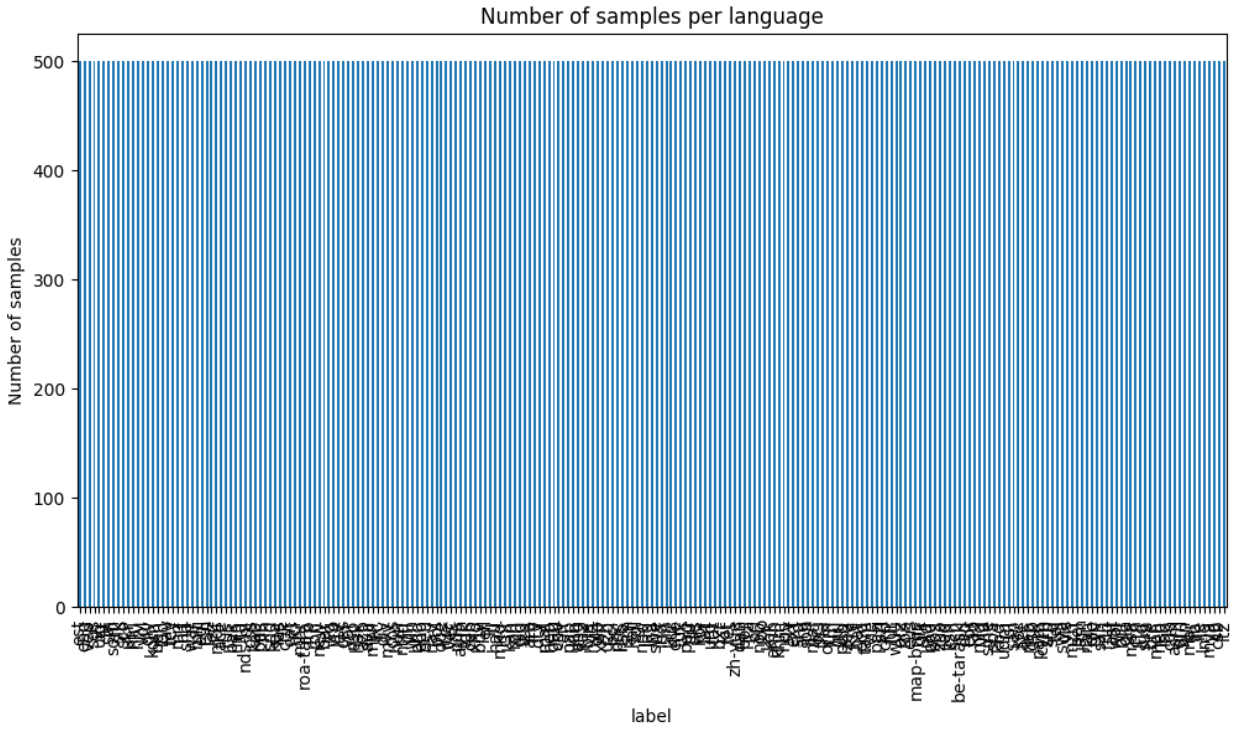
Problem Statement

The objective of this exercise was to make ourselves familiar with Linear models and Multi-Layer Perceptrons. Specifically, how they can be used in NLP tasks. The NLP task in the exercise was language identification from text sentences. In this report, we will discuss our approach, results and conclusions.

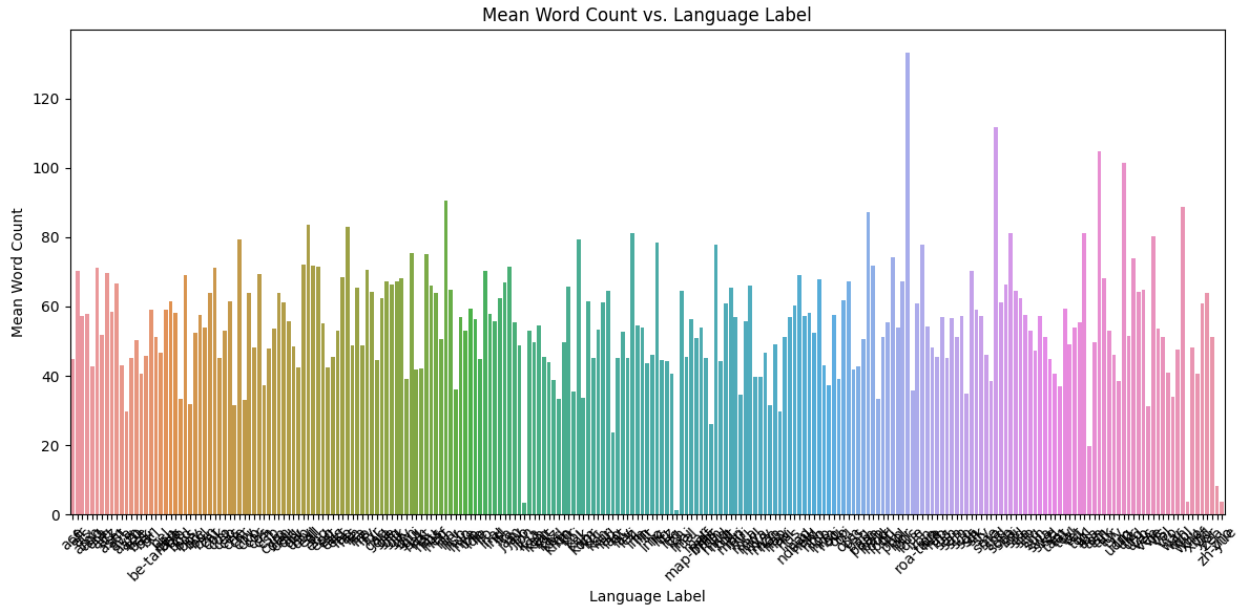
Exploratory Data Analysis (EDA)

We maintain the anonymity of the dataset in order to not exploit the test dataset and perform a brief Exploratory Data Analysis (EDA) on the train dataset.

The train dataset has 117500 samples (texts) that belong to 235 unique classes (languages) and all classes are balanced since all classes have 500 samples.



We also plot the word count averaged over all the samples in the classes to see if there is any variation across languages. As it appears, the average word count for languages varies a lot. This could be a potential additional feature for our models.



Data Preprocessing

We performed basic preprocessing on the combined dataset so that all the samples in both train and test are preprocessed. In our preprocessing we removed numbers, punctuation marks, excess white spaces and made all characters lowercase. In doing this we used the RegEx library.

We removed the numbers and the punctuation marks as they will not help in distinguishing between texts of different languages. We made all characters lowercase as we did not want our vectorizer to vectorize *Toffee* and *toffee* differently as it will only make the task more complex for the models.

Because of computational and time limitations we only work with 26 languages- [*'eng', 'deu', 'nld', 'dan', 'swe', 'nob', 'est', 'oci', 'zea', 'pnb', 'wuu', 'pan', 'fur','ton', 'glk', 'bod', 'jpn', 'srd', 'ext', 'sin', 'che', 'pag', 'als', 'koi', 'kir', 'bul'*].

Vectorizer and Label Encoder

After preprocessing our data, we used TF-IDF (Term Frequency - Inverse Document Frequency) vectorizer. TF-IDF uses frequency of a word or character to determine how relevant is that word or character to the document.

We vectorized our text using TF-IDF for both words and characters. We wanted to use the TF-IDF vectorization of characters as an additional feature along with TF-IDF vectorization of words. We choose the *ngram_range* parameter as (1,1), i.e., it considers one word or character for vectorization. While finding the optimal hyperparameters in grid search, we will experiment with this parameter.

We use Label Encoder from sklearn to encode our labels and to make them numeric for the models to be able to work with them.

Additional Features

In order to make our model more accurate and informed, we added two additional features to the existing vectorized words.

As discussed, one additional feature that we use are the vectorized characters. Another additional feature that we used is the average word length in the sample document. We do this because we

observe that in some languages, like Chinese, you do not require a lot of characters to make a sentence. Whereas, the same sentence in a language like English will take a lot of words.

We created a function that calculates this average word length. We then append this value to their corresponding row in the dataframe. And finally, we use this by using *FeatureUnion* and taking union of all three features.

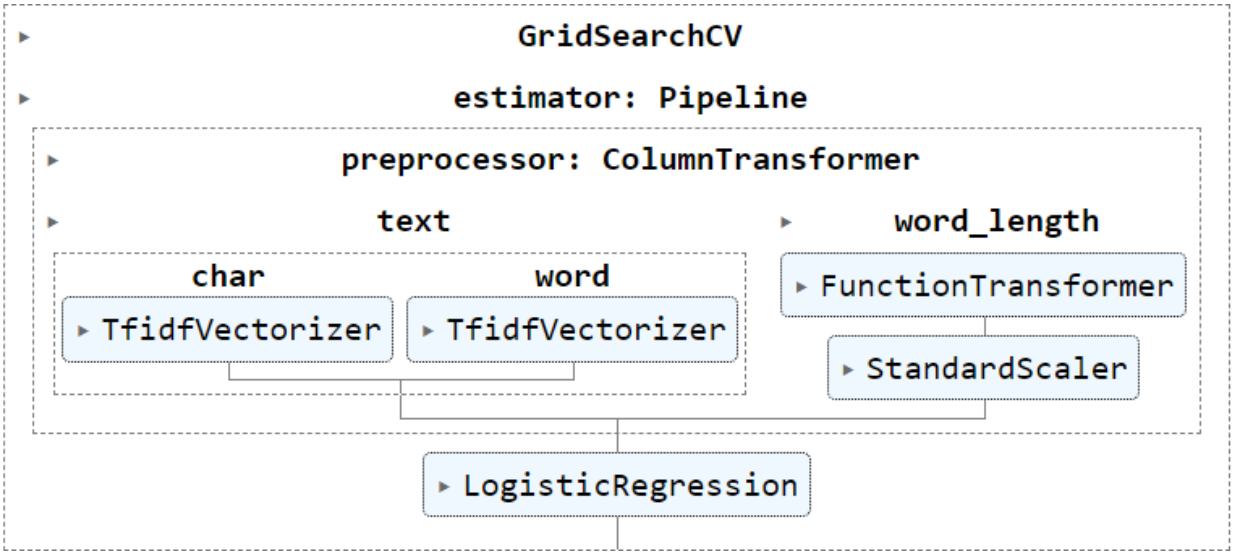
Pipeline Construction and Model Training

Logistic Regression

Pipeline Construction and Model Architecture

The pipeline has two parts-

1. Preprocessor: This part takes the feature union of all features and performs standard scaling on the *avg_word_length* feature.
2. Classifier: This part contains our Logistic Regression model that uses the *saga* solver and has the max iteration value to 20.



Model Training and Grid Search

To see how our model performs, we use *cross_val_score* from sklearn. Since we use 5 folds, the accuracies for the 5 splits are- 96.82%, 96.61%, 96.56% , 96.44%, 96.27%.

After ensuring satisfactory performance by the logistic regression model, we want to find the optimal hyperparameter set for our model. We choose the following parameters to undergo Grid Search-

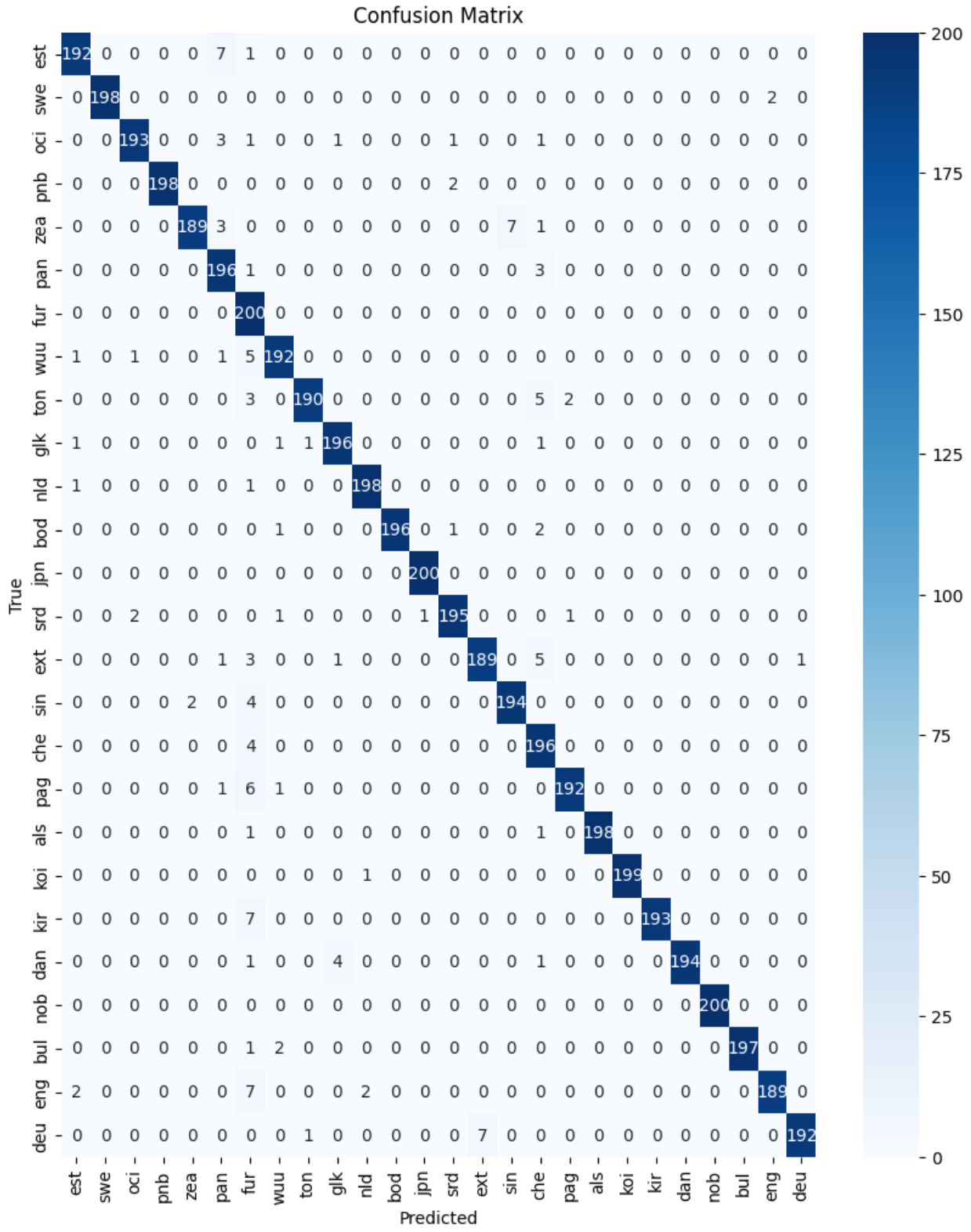
- *preprocessor__text__char__ngram_range*: [(1, 1), (1, 2)]
- *preprocessor__text__word__ngram_range*: [(1, 1), (1, 2)]
- *classifier__penalty*: ['l1', 'l2']
- *classifier__max_iter*: [20, 40]

After performing grid search for our experiment, we find the best hyperparameter set to be-

- classifier__max_iter: 40
- classifier__penalty: l2
- classifier__solver: saga
- preprocessor__text__char__ngram_range: (1, 2)
- preprocessor__text__word__ngram_range: (1, 1)

The train accuracy of the best performing model is 97.19% and the test accuracy is 97.42%

We also plot the confusion matrix of our model's predictions-


Confusion Matrix
True
Predicted
est swe oci pnb zea pan fur wuu ton glk nld bod jpn srd ext sin che pag als koi kir dan nob bul eng deu
200
175
150
125
100
75
50
25
0

Feature Importance using ELI5

Using the *ELI5* library, we explore the importance of each feature in the classification of its class. Below are the top ten most important features for the classes *ENG*, *SWE*, *NOB* and *JPN*.

	ENG	**SWE**	**NOB**	**JPN**
1	+4.209 word_the	+4.261 word_är	+3.867 word_og	+3.633 avg_word_length
2	+2.565 word_and	+3.799 word_och	+3.174 word_av	+2.968 char_の
3	+2.555 char_th	+3.022 char_ä	+2.471 word_ble	+2.098 char_に
4	+1.688 char_h	+2.188 char_r	+1.980 word_er	+1.802 char_、
5	+1.675 word_of	+2.114 word_av	+1.967 char_e	+1.724 char_る
6	+1.511 word_to	+2.078 char_ö	+1.931 char_k	+1.667 char_一
7	+1.495 word_was	+2.047 char_är	+1.801 word_som	+1.612 char_た
8	+1.460 word_in	+1.910 char_å	+1.756 word_til	+1.589 char_は
9	+1.427 char_o	+1.692 word_den	+1.729 char_ø	+1.490 char_と
10	+1.403 char_d	+1.336 char_ör	+1.619 char_t	+1.347 char_で

Ablation Study

We conducted an ablation study where for the two classes for which our model performed the best, i.e., *JPN* and *NOB*, we used a limited number of characters per training and testing

experiment. In the first experiment we used 1600 characters, 500 in the second and 100 in the third.

Experiment	**Character Limit**	**Accuracy Score**
Experiment 1	1600	98.75%
Experiment 2	500	98.75%
Experiment 3	100	99%

We concluded that these results do not resemble the expected results, which were supposed to be a decreasing accuracy with decreasing amount of data available for training, because-

1. In addition to the vectorized words, we also use vectorized characters and average length of words as features which vary immensely for the two classes.
2. Because of such a high difference in the features, the model will always excel at distinguishing the two classes.

We repeated the same experiment with 1600, 100 and 5 character limits and observed the same results. This strengthens our claim of the abnormal observation.

Multi-Layered Perceptron

Pipeline Construction and Model Architecture

The preprocess pipeline is the same as before where it combines the *word* and *char* TF-IDF vectorizer along with the additional feature *avg_word_length* which is the average length of the words in a document sample.

We then create the model in torch that has-

1. 2 layers (Input and Output)

2. Input Dimension: 201115
3. Output Dimensions: 26
4. Hidden Dimension (No. of hidden neurons: 100)
5. Activation Functions: ReLU()
6. Dropout Probability: 0.2, after the first layer

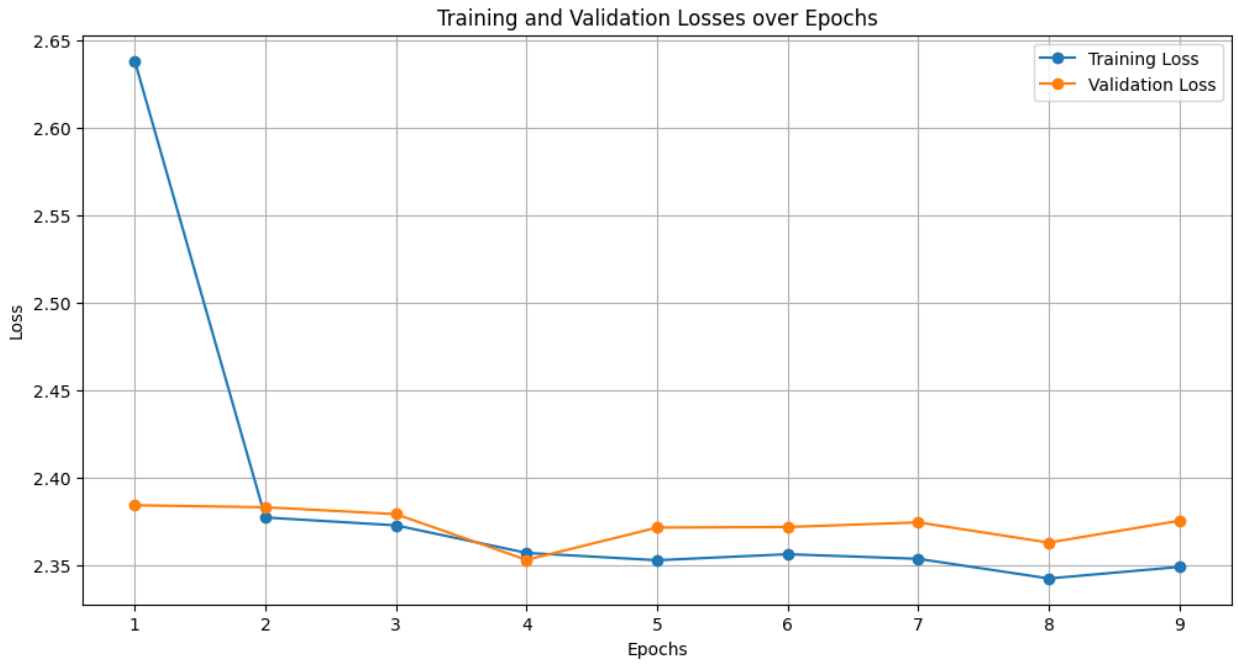
Model Training

We train our pipeline that consists of the preprocesses and the model. We include *early stopping* (with *patience = 5*) in our process as we do not want our model to overfit. For the same reason, we also have included dropout after the first layer in our model architecture.

For training-

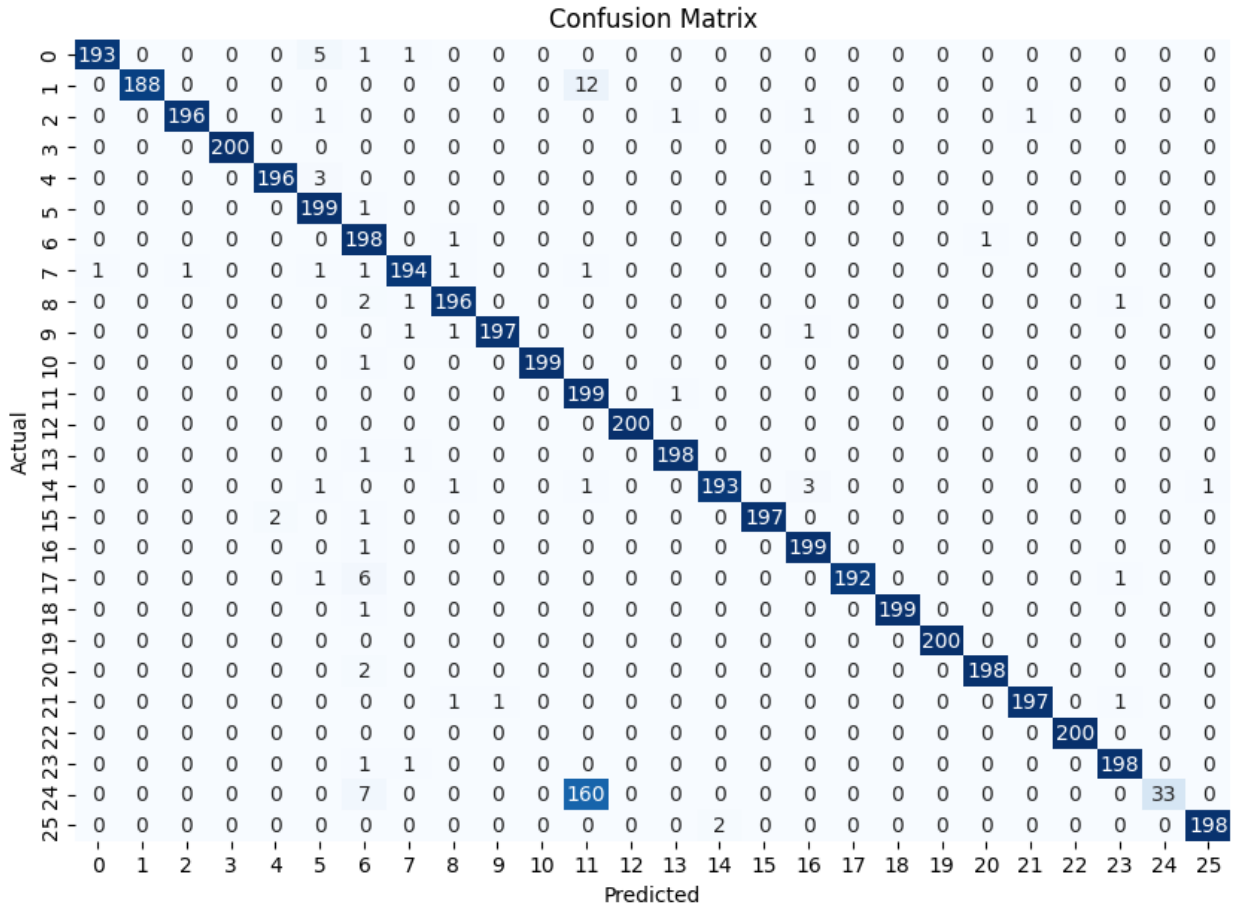
1. We use the loss function *nn.CrossEntropyLoss* since we have categorical data.
2. Number of max epochs: 10
3. Learning Rate: 0.01
4. Optimizer: Adam

We wrap the model and these training hyperparameters using the *NeuralNetClassifier* from *skorch*.



Testing the Pipeline

On the test set, our model achieved an accuracy of 95.33%. And we obtained the following confusion matrix for our predictions-



Since the model achieved 95%+ accuracy in testing, we did not opt to use *Grid Search* for finding the optimal hyperparameters.