Language Technology - EDAN20
Assignment 3 - Fall 2020

A simple language classifier
Multi-layer Perceptron (MLP) - Scikit-learn

Hicham Mohamad, hi8826mo-s

November 8, 2020

1 Introduction

In this assignment, we will apply and explore the supervised learning algorithm **Multi-layer Perceptron** (MLP) for building a language classifier using **Scikit-learn**. We will consider **Tatoeba dataset** limited to the three languages only: French (fra), English (eng), and Swedish (swe). The resulting **language detector** is inspired from Google's Compact Language Detector version 3, also called CLD3 [8].

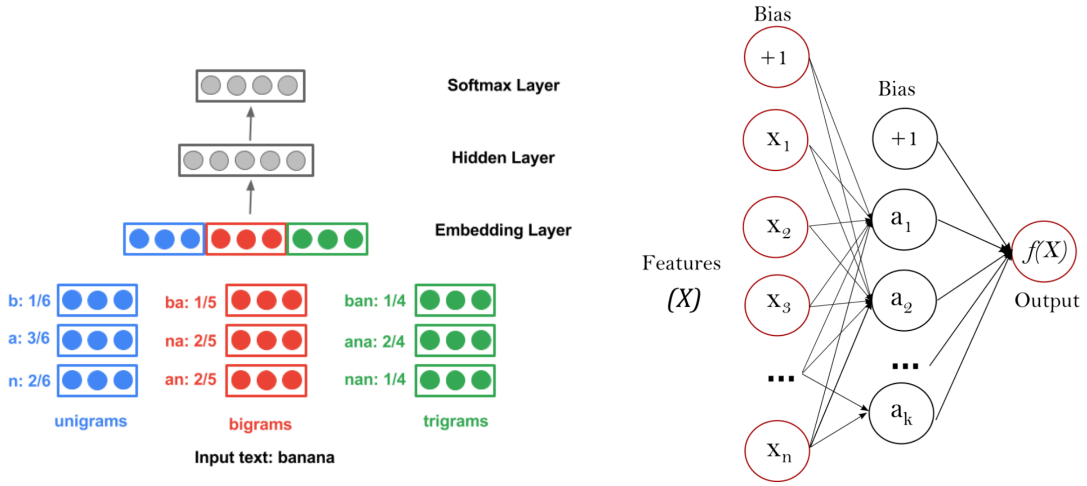

*Figure 1: a) **CLD3 overall architecture**: CLD3 is a neural network model for language identification. As an example, if the input text is "banana", then one of the extracted trigrams is "ana" and the corresponding fraction is 2/4. The model averages the embeddings corresponding to each ngram type according to the fractions. b) One hidden layer MLP.*

2 CLD3 System Overview

CLD3 is a **neural network model** for language identification, which can be summarized by the following:

- The inference code consists of these principal steps:
  - The character **ngrams** are extracted from the input text.
  - It computes the **fraction of times** each of these ngrams appears.
  - The ngrams are **hashed** down to an id within a small range.

  - Each id is represented by a **dense embedding vector** estimated during training.

- The **averaged embeddings** are concatenated to produce the **embedding layer**.
- The remaining components of the network are a hidden (Rectified linear) layer and a softmax layer.
- After performing a **forward pass** through the network, we get a language prediction for the input text.

Understanding the X matrix (feature matrix)

In this section, we investigate the CLD3 features, by considering only three languages. The CLD3's original description uses **relative frequencies**, i.e. counts of a letter divided by the total counts of letters in the text. Below an excerpt of Tatoeba dataset is illustrated:

```
1276    eng     Let's try something.
1277    eng     I have to go to sleep.
1280    eng     Today is June 18th and it is Muiriel's birthday!
...
1115    fra     Lorsqu'il a demandé qui avait cassé la fenêtre, tous les garçons ont pris
un air innocent.
1279    fra     Je ne supporte pas ce type.
1441    fra     Pour une fois dans ma vie je fais un bon geste... Et ça ne sert à rien.
...
337413  swe     Vi trodde att det var ett flygande tefat.
341910  swe     Detta är huset jag bodde i när jag var barn.
341938  swe     Vi hade roligt på stranden igår.
...
```

From the three languages example above, (French fra, English eng, and Swedish swe), we need to create manually a simplified $\mathbf{X}$ matrix where we represent the 9 texts with CLD3 features. However, here we use a restricted set of features and consider only the **letters** a, b, and n and the **bigrams** an, ba, and na. Thus, by using the **raw counts**, we create the (9×6) $\mathbf{X}$ matrix where each column contains these counts: `[ #a, #b, #n, #an, #ba, #na ]`.

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 1 & 0 & 0 \\ 8 & 0 & 8 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 4 & 1 & 6 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 & 0 \\ 5 & 2 & 2 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} \text{eng} \\ \text{eng} \\ \text{eng} \\ \text{fra} \\ \text{fra} \\ \text{fra} \\ \text{swe} \\ \text{swe} \\ \text{swe} \end{bmatrix} \tag{1}$$

3 Features extracting - Building X and y

The CLD3 architecture uses **embeddings**. In this task, we will simplify it and we will use a feature vector instead consisting of the **character frequencies**. In this way, we start with a **baseline** which corresponds to the minimal technique that is used to assess the difficulty of the task and to compare with further programs. Thus, in the implemented Python code we:

1. Extract the character **frequency dictionaries** from `dataset_small` corresponding to its 3rd index and set them in a list. In this way, we poduce a new list of data points with the **unigrams** only.
2. Convert the list of dictionaries into an $\mathbf{X}$ matrix using `DictVectorizer` class from the **scikit-learn** library [5], which has method `fit_transform()` to convert dictionaries into such vectors.
3. Extract the list of **language symbols** `fra, eng, swe` from the generated list of dictionaries `dataset_small_feat`.

4. Build two **indices**, dictionaries called `lang2inxand` and `inx2lang`, for mapping the symbols to integers and the integers to symbols.
5. Convert the list of symbols into a **numerical vector** called **y** vector.

4 Building the model

Multi-layer Perceptron MLP

As illustrated in Figure 1b, multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f(.) : \mathbb{R}^m \rightarrow \mathbb{R}^o$ by training on a dataset, where m is the number of dimensions for **input** and o is the number of dimensions for **output**. Given the set of features **X** and a target **y** , it can learn a non-linear function approximator for either **classification** or regression.

MLPClassifier in Scikit-learn

In this task, we need to use `MLPClassifier` in Scikit-learn for building our classifier `clf`. MLPClassifier trains iteratively since at each time step the partial derivatives of the **loss function** with respect to the model parameters are computed to update the parameters. It can also have a **regularization** term added to the loss function that shrinks model parameters to prevent **overfitting**.

Here we describe the different experiments and steps to achieve an decent performance:

1. In the begining, we create a baseline neural network with a hidden layer of 50 nodes and a relu activation layer. For the parameters, we set the maximal number of iterations to 5, verbose to True and the default values for the rest.
2. We shuffle the indices and split the dataset into **training** and a **validation** sets.
3. Fit the model on the training set, and we get loss = 0.0598 after 5 iterations.
4. We predict the languages of the **X** features in the validation dataset, and we obtain accuracy 0.9788.
5. We evaluate the model using **F1 score** and get the following results

```
              precision    recall  f1-score   support

         fra       0.97      0.95      0.96     87715
         eng       0.98      0.99      0.99    273517
         swe       0.97      0.89      0.93      7613

avg / total       0.98      0.98      0.98    368845

Micro F1: 0.9788203717008499
Macro F1 0.9583552353995707
```

we can also use **confusion matrix**

```
array([[ 83759,   3915,     41],
       [  2852, 270482,    183],
       [    77,    744,   6792]], dtype=int64)
```

We could tune the parameters and increase the number of iterations to improve the score, but with this classifier we already obtain good performance.

5 Predicting

In this task, we need to test the classifier and predict the languages of the following strings

docs = ["Salut les gars !", "Hejsan grabbar!", "Hello guys!", "Hejsan tjejer!"]

To solve this problem, we need to

1. Create features vectors from this list.
2. Use the classifier implemented above and run the prediction.

The obtained predicted languages are ['fra', 'swe', 'eng', 'swe']

6 Appendix

References

[1] Pierre Nugues, *Language processing with Perl and Prolog*, 2nd edition, 2014, Springer.

[2] P. Nugues, Language Technology - EDAN20, Lectures notes: `https://cs.lth.se/edan20/`

[3] D. Jurafsky, J. Martin, *Speech and Language Processing*, 3rd edition. Online: `https://web.stanford.edu/~jurafsky/slp3/`

[4] Pierre Nugues, Assignment 3: `https://github.com/pnugues/edan20/blob/master/notebooks/3-language_detector.ipynb`

[5] Scikit-learn, Feature Extraction: `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html`

[6] Scikit-Learn, Neural Network Models `https://scikit-learn.org/stable/modules/neural_networks_supervised.html`

[7] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[8] CLD3 - Compact Language DEtector v3: `https://github.com/google/cld3`

[9] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`