Topic classification using pseudo-labeling technique

Machine Learning Project

by

Eduardo Calò, Thibo Rosemplatt

under the guidance of

Marianne Clausel

March, 2020

Contents

1 Technique overview **1**
1.1 Introduction 1
1.2 Semi-supervised learning and Pseudo-labeling 1
1.3 Mathematical background 2

2 Dataset **5**
2.1 Goal 5
2.2 Choice of the dataset 5
2.2.1 Previous attempts 5
2.2.2 Final choice 6

3 Code overview **7**

4 Results and discussion **10**

Bibliography **12**

Chapter 1

Technique overview

1.1 Introduction

The purpose that fueled our choice for pseudo-labeling technique has been the fact that acquiring large amount of data needed for a Machine Learning (ML) problem to be solved incurs in various difficulties. For example, skilled experts are always needed for labeling samples. As known, these tasks are expensive and time-consuming, since large amount of time and labor force are required. Thus, the cost associated with the labeling process may render large, fully labeled training sets infeasible, whereas acquisition of unlabeled data is relatively inexpensive.

In order to partially avoid labor intensive, expensive, time-consuming and error-prone tasks, one of the best trade-off is to use a small amount of human annotated data, together with a large amount of unlabeled data. This is what is known as Semi-supervised learning (SSL) approach. In our project, in particular, we are exploiting the pseudo-labeling technique.

1.2 Semi-supervised learning and Pseudo-labeling

In this section, we will shortly describe SSL, and give a more detailed overview on how pseudo-labeling technique works.

Semi-supervised learning (see Zhu [2005] for a comprehensive review) is an approach to Machine Learning that combines a small amount of labeled data with a large amount of unlabeled data, during training. As the name suggests, SSL falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data).

Unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. In such situation, SSL can be of great practical value. SSL is also of theoretical interest in ML and as a model for human learning.

In particular, in our project, we have used the pseudo-labeling technique. In this technique, instead of manually labeling the unlabeled data, approximate labels are being given to the unlabeled data, on the basis of the labeled data.

First proposed by Lee [2013], this method tries to improve a model's performance just by looping through the following 4 basic steps:

1. Train model on a batch of labeled data.
2. Use the trained model to predict labels on a batch of unlabeled data.
3. Use the predicted labels to calculate the loss on unlabeled data.
4. Combine labeled loss with unlabeled loss and backpropagate.

Since we decided to work using conventional ML algorithms, pseudo-labeling needs to be used in a slightly different way, although the original concept is maintained. The steps are the following (refer to Figure 1.1 for a visualization of these steps[1], and to Diagram 1.2 for the algorithm[2]):

1. First, a classifier is trained on our labeled train set.
2. Next, this classifier is used to predict the labels on a randomly sampled batch from the unlabeled dataset.
3. Both the original train set and the predicted set are combined, and retraining of the classifier on this new dataset is performed.
4. Repeat steps 2 and 3 until all of the unlabeled data have been used.

This technique is usually implemented thanks to a wrapper, under the form of Class[3]. A detailed explanation of the Class we have used will be given in Section 3.

1.3 Mathematical background

As seen earlier, the training part of the pseudo-classification is performed following several steps. At each step, the key is the computation of the loss function which will allow to adjust the parameters of the model (back-propagation). The loss is calculated as follows (Lee [2013]):

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L(y_i^m, f_i^m) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L(y_i'^m, f_i'^m)$$

where n is the number of mini-batch in labeled data for stochastic gradient descent,

[1] https://www.analyticsvidhya.com/blog/2017/09/pseudo-labelling-semi-supervised-learning-technique/

[2] https://towardsdatascience.com/pseudo-labeling-to-deal-with-small-datasets-what-why-how-fd6f903213af

[3] https://github.com/anirudhshenoy/pseudo_labeling_small_datasets/blob/master/pseudo_label-Logistic_reg.ipynb

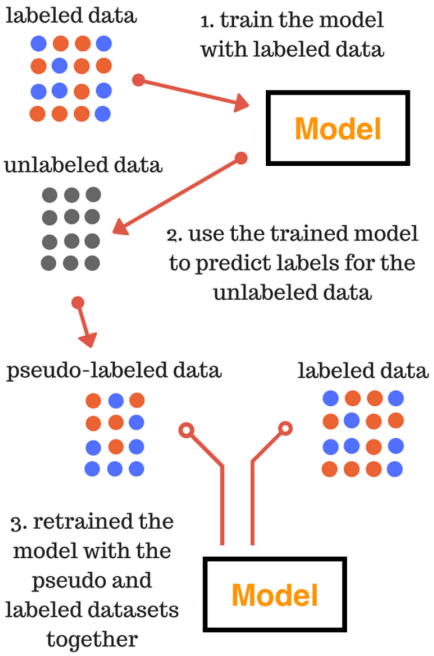

Figure 1.1: Pseudo-labeling workflow.

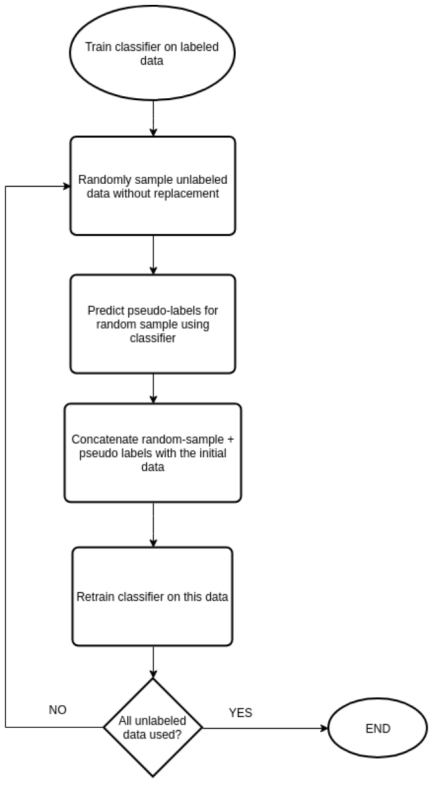

Figure 1.2: Pseudo-labeling algorithm for conventional ML.

n' for unlabeled data, f_i^m is the output units of m's sample in labeled data, y_i^m is the label of that, $f_i'^m$ for unlabeled data, $y_i'^m$ is the pseudo-label of that for unlabeled data.

$\alpha(t)$ is a coefficient used to control the contribution of unlabeled data to the overall loss. It increases over time during the training. It is defined as follows:

$$\alpha(t) = \begin{cases} 0 & \text{if } t < T_1 \\ \frac{t-1}{T_2 - T_1} * \alpha_f, & \text{if } T_1 \leq t \leq T_2 \\ \alpha_f, & \text{if } T_2 \leq t \end{cases}$$

α_f, T_1 and T_2 are parameters that have to be adjusted with respect to the problem that is attempted to be solved, and the dataset that is used. By default, we have $\alpha_f = 3$, $T_1 = 100$ and $T_2 = 600$.

To put this it into words, the formula can be expressed as:

$$\textit{Loss per batch} \;=\; \textit{Labelled loss} \;+\; \textit{weight} \;*\; \textit{unlabelled loss}$$

The iterations will stop when all the unlabelled data are pseudo-labelled. Eventually, if the number of epochs allows it, the whole process will be repeated, leading the classifier to be more efficient.

Chapter 2

Dataset

2.1 Goal

We have decided to apply the pseudo-labeling technique to binary topic classification[1], trying out on several different datasets. In Section 2.2, we will list all the datasets we have tried to exploit, before finding the one that fit our scope. Basically, we have been dealing with a binary classification ML problem in which there is an attempt to estimate the mapping function (f) from the input variables (X, texts that will be pre-processed and vectorized) to discrete or categorical output variables (y, two classes).

2.2 Choice of the dataset

2.2.1 Previous attempts

Our first attempt was to use the SMS Spam Collection dataset[2]. Our aim was to train a classifier on few labelled data, which in the end could predict for a given text message whether it was a spam or not. In this configuration, having the regular supervised learning as a baseline, the pseudo-labelling technique did not improve (or even lowered) the accuracy of our classifier. The total amount of data was actually few, this might be a reason why the pseudo labelling failed.

Our second attempt was to use open data from the French government. France is indeed one of the leader countries regarding open data accessibility[3]. Among the available data sets, we went for "Questions écrites" dataset. It gathers all the written questions to the government asked the French Members of Parliament since 2012[4]. Questions to government are questions asked by MPs to the government in the parliament on current issues. Our aim was to create a classifier which given

[1] https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f

[2] https://www.kaggle.com/uciml/sms-spam-collection-dataset

[3] https://opendatabarometer.org/2ndEdition/analysis/rankings.html

[4] https://data.gouv.fr/fr/datasets/questions-ecrites

an input text written by an MP, would recognize the political party to which this MP belongs. This attempt was either not successful: even the supervised classifier trained with a majority of labelled data was not really accurate, no need to say that the pseudo-labeler did not help. This time, we think that the low results are due to the nature of the problem: guessing a political party from a text is also a really difficult task for humans.

In the next subsection, the choice that seems to be the best for illustrating the pseudo-labeling technique will be introduced.

2.2.2 Final choice

The dataset which has finally given us the expected results is Amazon Review Data[5]. The whole corpus contains millions of reviews from customers over items bought on Amazon. Each entry is made of multiple features (Review content, Reviewer ID, Item ID, Purchase date, category of the item...). Our idea has been to create a classifier that given a text review, would predict the category of the item to which the review is addressed. For this purpose, the only feature we need to keep is the review content, and the target was the category of the reviewed item. However, for computational complexity concerns, we decided to constrain ourselves to 2 categories, and thus deal with a binary classifier. We have chosen "Luxury Beauty" (34,278 reviews) and "Software" (12,805 reviews) subsets, as they come from two totally different realms, which might help the classifier distinguish the items better. The following Chapter will give details on how the dataset was processed.

[5] `https://nijianmo.github.io/amazon/index.html`

Chapter 3

Code overview

In this section a detailed, step-by-step description of the implementation of the code is provided.

If you want to run the code provided, you do not have to go through the linguistic-processing. You can simply upload the pickles provided in addition to this document. dataframe.pkl is a shortcut to skip the linguistic preprocessing. best_parameters.pkl allows to skip the search of the best set of parameters.

The first step, as usual, is to **import libraries**. For this project, we have used many common libraries for ML and linguistic processing, such as pandas[1], NumPy[2], spaCy[3], NLTK[4] and Scikit-learn[5].

The second step is to **load data** from the JSON files (see Section 2.2.2 for a description of the dataset) and extract the attributes needed, i.e. the actual reviews categories of the products. We decided to pad to 10,000 samples for each category, so that enough data, for both training and testing, would be available, and above all, to prevent fairness issues. Indeed, without padding, the data provided would be highly biased, as the classifier would be overly exposed to "Luxury Beauty" reviews. Moreover, we discarded the JSON files which are ill-formatted and created a DataFrame with texts and labels.

Then, we performed a **linguistic pre-processing** using spaCy. Each review undergoes a process of tokenization, PoS tagging and lemmatization. Stopwords, punctuation, pronouns, and symbols are removed. A clean text for each review is returned and the DataFrame updated with clean texts.

Thereafter, comes the **data preparation** part, in which the train and test data sets are created and the features extracted. When creating the data sets, we split

[1] `https://pandas.pydata.org/`
[2] `https://numpy.org/`
[3] `https://spacy.io/`
[4] `http://www.nltk.org/`
[5] `https://scikit-learn.org/stable/`

using a ratio of 80-20. 80% of the data have been used for training, and the remaining 20% for testing. We have further divided the training data set to pretend that 60% of it were composed of unlabeled data. So, we ended up with using 6400 labeled samples, 9600 unlabeled samples, and 4000 samples for testing.

For **feature engineering**, term frequency-inverse document frequency (TF-IDF) vectorization technique has been used. TF-IDF allows to represent each text in the data set with a vector (with a number of dimensions corresponding to the number of features one decides to keep; 9000 in our case), and the whole data set as a matrix (where each row is a text in the data set, and the columns are the features). Terms that occur frequently across all documents are weighed down. Furthermore, we have decided not to lowercase, since we believe that upper-cased names of products and software can help the algorithm learn better. The 3 parts of the data set have been vectorized in this step.

The following step is dedicated to the **choice of the classifier**. In order to find out which model could perform better, we have decided to make a baseline estimation on the labeled data, comparing 4 models usually used for classification (Random Forest Classifier, Linear Support Vector Classification, Multinomial Naive Bayes, and Logistic Regression), and using 5-folds cross-validation (see Figure 3.1).

Figure 3.1: Cross validation with 5 folds.

Thanks to this baseline estimation, we found out that the model which performs better is Multinomial Naive Bayes, which, applied only on the labeled data, as shown in Figure 3.2, performs with 97.61% of accuracy. We proceeded then with a normal submission using MultinomialNB (refer to Chapter 4 for further details and results).

The following block is devoted to the **implementation the pseudo classifier**, under the form of a Class. The first part of the Class is dedicated to the definition of the initial attributes: the model used in the pseudo-labeling, the unlabeled data, the sample rate (percentage of unlabeled data taken in each iteration), 2 thresholds (upper and lower), *unlabelled_indices* (which creates a list of all the indices of the

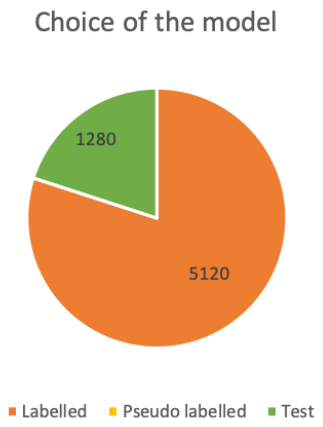

Figure 3.2: Data distribution for the choice of the model.

unlabeled samples), *sample_size* (which is the number of rows from the unlabeled data to sample in each iteration, based on the sample rate given as parameter). At this moment, the indices are shuffled, so that each time the unlabeled data are randomly chosen. The *__pop_rows* function takes a certain number of rows (based on sample size) from the shuffled list of indices and removes these chosen rows from the list of indices (so that they will not be used in the next iteration).

The *fit* method is the main part, where the pseudo-labeling is actually done, starting from the original labeled batch. The number of iterations (based on the number of unlabeled data and the sample size) is set here. A *for loop* over the number of iterations is initialized. During each iteration, the model is fit using the labeled data (in each iteration the number of these labeled data will increase, since new pseudo-labels will be given to a batch of unlabeled data), and a new batch of unlabeled samples is taken (*chosen_unlabelled_rows*). Thereafter, the model makes predictions over these unlabeled samples (*pseudo_labels_prob*). At this moment, the code will first find the maximum probability, and then, find the rows which are within our threshold values (*labels_within_threshold*). The default values of these threshold are 0,4 (for *lower_threshold*) and 0,6 (for *upper_threshold*). Argmax[6] method is then used to find the category with the highest probability. Once these pseudo-labels have been given, the data are combined (using *vstack* method from NumPy), creating the new X and y for the new iteration.

In the penultimate section of the code, we tried to **find the best combination of hyper parameters** of the pseudo classifier class. To tune them we have used a nested triple *while loop*, so that all the combinations were tested, saving all the results in a DataFrame for easy retrieval. We finally found out that the best combination is :

```
sample_rate = 0.160, lower_threshold = 0.200, upper_threshold = 0.500
```

We then finally obtain the **results**, which will be discussed in Chapter 4.

[6] `https://en.wikipedia.org/wiki/Arg_max`

Chapter 4

Results and discussion

This chapter will mention the results obtained by the pseudo-classifier that we implemented, and will show that the pseudo-labelling approach was successful to improve the accuracy of the classification.

A good way to judge the abilities of our classifier is to compare its results with the ones of a normal submission, i.e. coming from a classical supervised learning algorithm based on the same model (Multinomial NB). To perform this comparison, we had to carefully train the two classifiers under the same conditions. For this interest of fairness, the same proportion, amount, and batch of labelled and test data have been used in both cases.

The only difference between the two setups is the unlabelled data: in the normal submission they are not used at all. (see Figure 4.1 and 4.2).

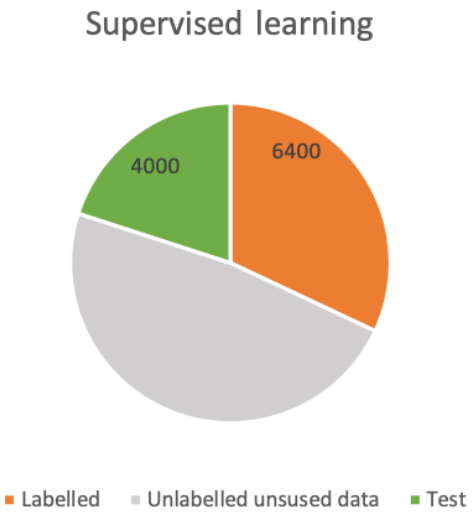

Figure 4.1: Data distribution for the normal submission.

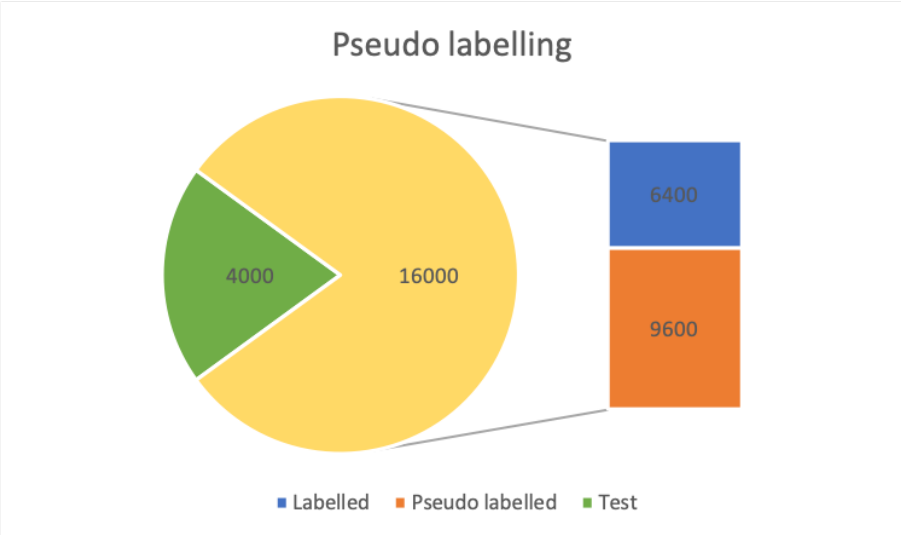

Figure 4.2: Data distribution for the pseudo labelling.

In the first scenario, the regular supervised algorithm performs with an accuracy of **72,03%**. In the second scenario, the pseudo-classifier reaches **76.18%** of accuracy.

We can observe an improvement of around 4% when using the pseudo-labelling method. Such improvement is considerable in real-life situations as it could save a lot of money and time, avoiding investing too much in human-annotation tasks. So, in our experiment, we have shown that pseudo-labelling can remarkably enhance the performance of a classifier, when the available annotated data is sparse.

Bibliography

Xiaojin Jerry Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2005.

Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 2, 2013.