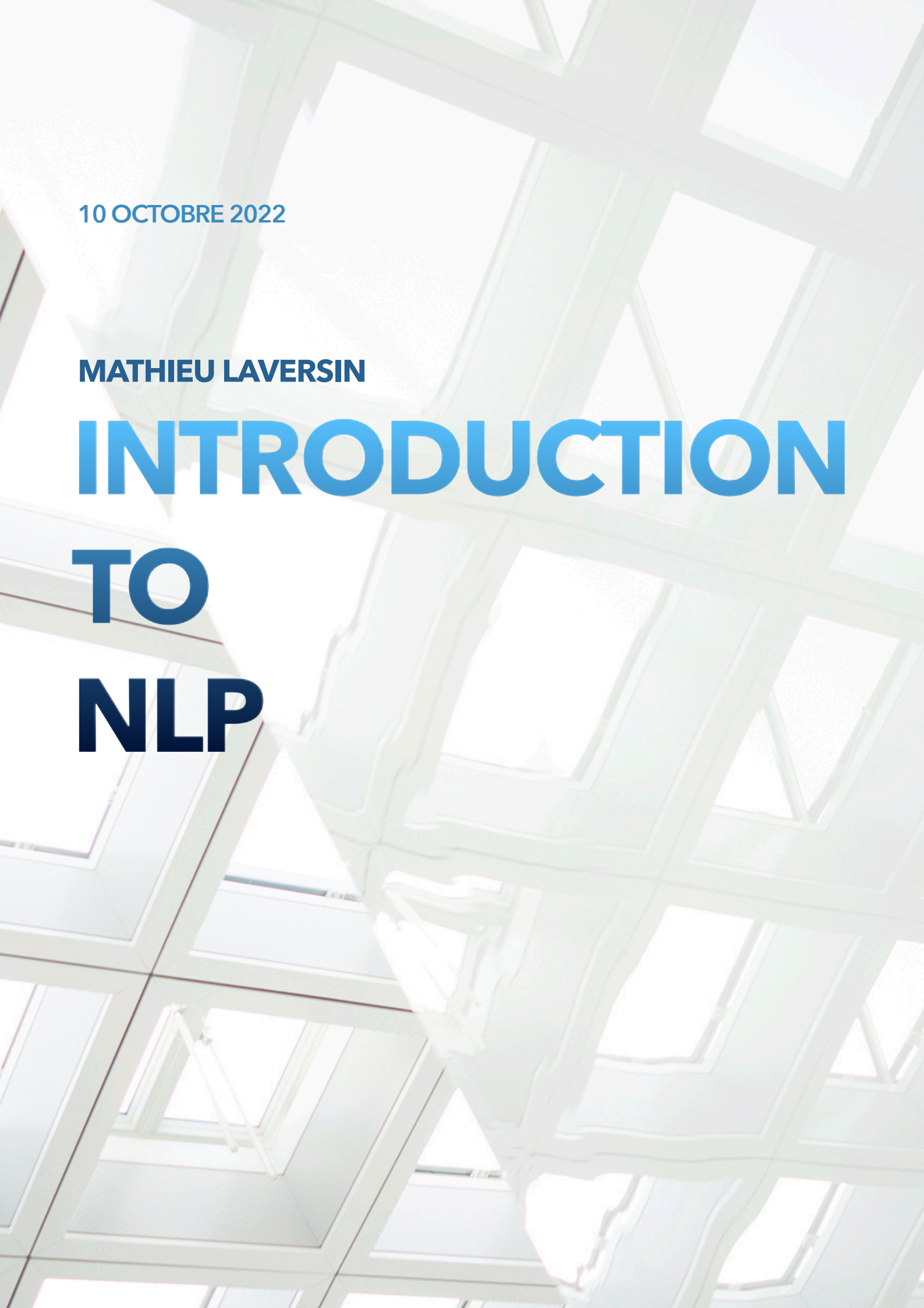


10 OCTOBRE 2022

MATHIEU LAVERSIN

INTRODUCTION TO NLP

Sentiment Analysis using Python

LOGISTIC REGRESSION

1. Introduction (definition, goals, preprocessing)

In a first time, we will define what is NLP. It refers to the branch of computer science and more specifically of AI. It will give the ability to the computers to understand text and spoken word as much as the human do.

Goals of this TP and what I learned : I learned how to mix the human language and the computer learning, how you can train with logistic regression the computer to categorise the texts in different way and at the end predict your own text. The more incredible was to use a Twitter API to do those tasks. We understood in this way the importance of NLP.

Before doing anything we needed to import libraries (nltk for Twitter, os for computer vision and the classic one to play with the data).

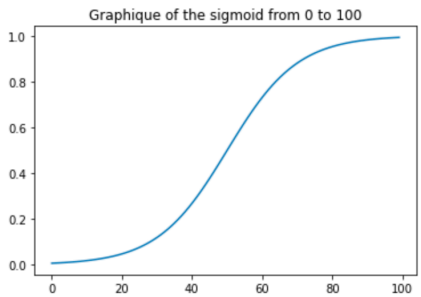
After we prepare the data :

- 1) Split with positive and Negative

-2) Train test split it, to have training set and testing set.

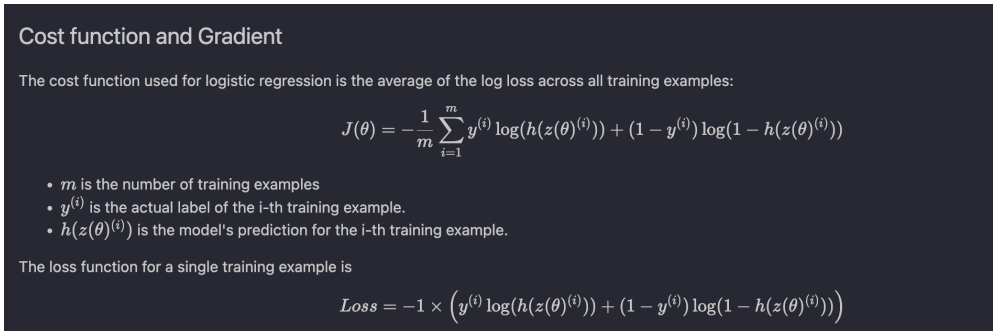
After we create the frequency dictionary by the build_freqs() : This counts how often a word in the 'corpus' (the entire set of tweets) was associated with a positive label 1 or a negative label 0. It then builds the freqs dictionary, where each key is a (word,label) tuple, and the value is the count of its frequency within the corpus of tweets.

We preprocessed the data by using : process_tweet() it splits the tweet into words, removes stop words and applies stemming. This is also called tokenization.

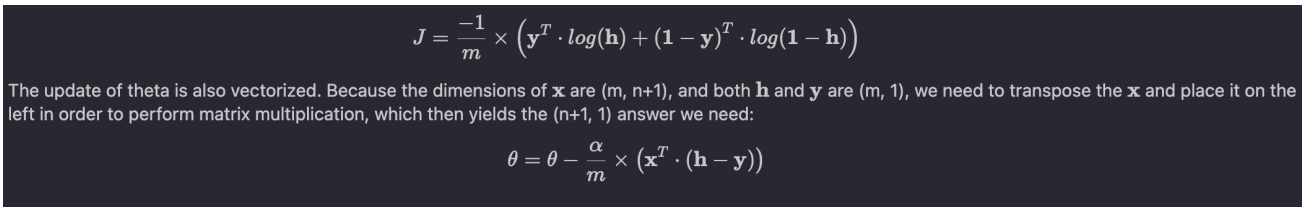


First of all we had to implement the sigmoid function which was really simple with numpy. A little test was implemented to verify our function.

There was a little reminder about the cost function and the gradient for logistic regression.



And after a little one on gradient descent function.



We implemented it on python, without forgetting to use (np.dot), at the end of the implementation we had a cost after training of 0.67 and a vector of Weights.

2. Logistic Regression (extracting features in a text, implementing logistic regression, logistic regression and test it for a NLP example and analyse the error).

Extracting features : by using process_tweet() we implemented the function to extract the features. Given a list of tweets, we extract the features and stored it in a list (2 features). After we train our logistic regression classifier on the features and test the classifier on a validation test.

Little comments on the def extract_features(tweet, freqs) : It wasn't too difficult to implement it but using freqs.get() to increment the positive label 1 or 0 take me a little bit time.

It was the time to training our model after extracted the features, the steps was to apply the gradient descent and compute the cost and the vector of weights.

When it was done, the last step was to test our logistic regression on some new input that our model has not seen before (with predict_tweet()). The testing was really cool :

```python
for tweet in ['I am happy', 'I am bad', 'this movie should have been great.', 'great', 'great great', 'great great great', 'great great great grea
    print( '%s -> %f' % (tweet, predict_tweet(tweet, freqs, theta)))
```

Python

```
I am happy -> 0.518580
I am bad -> 0.494339
this movie should have been great. -> 0.515331
great -> 0.515464
great great -> 0.530898
great great great -> 0.546273
great great great great -> 0.561561
```

When we add some « positive word » the score grows but not really, we tested the « high words » and the smiley :) is really powerful (0.90).

Checking performance using the test set :

After trained our model using the training set, we have to check how the model perform on real, unseen data by testing it on the test set. The condition of the function was quite simple, if the y predicted has a score upper than 0.5 it was positive then negative. And we computed the accuracy obviously.

To finish this TP we compute the error analysis to check the misclassifications, the common this thing is that our dictionary and classification test is too weak to classified correctly those tweets, we should needed to training it a little bit.

Before tweet anything now I should use my own function and test some sentence to see if I'm a kind or mean person :)

Sentiment Analysis using Python

NAIVE BAYES

In this Second TP, we'll use naive Bayes for sentiment analysis always on tweets. I don't have to re explain the libraries, it's quote the same one. And we'll also use the process_tweet() to cleans the text, tokenizes it into separate words, removes stopwords, and converts words to stems. And lookup() to returns the number of times that word and label tuple appears in the collection of tweets.

What is Naive Bayes : This is a simple (naive) classification method based on Bayes rule. It relies on a very simple representation of the document (called the **bag of words** representation)

Hint : P(c|x) = P(x|c) * P(c) / P(x) it called the bayes theorem.

After removing the noise we have to implement the helper function it will be useful to train our model. We created a dictionary were the keys were a tuple composed by word and label and the values were corresponding to a value. It took the result, the tweets and a list corresponding to the sentiment of each tweet (0 / 1).

Example : { (rather »,1) : 2} Means that the word rather appears 2 times in the tweet with the label 1 positive

Testing:

```python
# Testing your function
result = {}
tweets = ['i am happy', 'i am tricked', 'i am sad', 'i am tired', 'i am tired']
ys = [1, 0, 0, 0, 0]
count_tweets(result, tweets, ys)
```

Python

```
{('happi', 1): 1, ('trick', 0): 1, ('sad', 0): 1, ('tire', 0): 2}
```

The training of the Naive Bayes Model :

Good model in terms on complexity and ecological, quite fast to launch. To not repeat the Clear explications of the notebook, here's the screen of the explanation.

So how do you train a Naive Bayes classifier?

- The first part of training a naive bayes classifier is to identify the number of classes that you have.
- You will create a probability for each class. $P(D_{pos})$ is the probability that the document is positive. $P(D_{neg})$ is the probability that the document is negative. Use the formulas as follows and store the values in a dictionary:

$$P(D_{pos}) = \frac{D_{pos}}{D} \tag{1}$$

$$P(D_{neg}) = \frac{D_{neg}}{D} \tag{2}$$

Where D is the total number of documents, or tweets in this case, D_{pos} is the total number of positive tweets and D_{neg} is the total number of negative tweets.

It would be too long to explain each parameters so we computed the frequency of positive and negative, the number of positive word and negative. After the number of positives/ negative tweets. Used the log prior and computed the log likelihood.

-> The log likelihood deserves an explanation, it's a story of equality of the scale, without log it would be [0;1[negative and the other scale would have been positive. With the log it's scaled to -/+ infinite. Also the product would have tend 0 if it's the same reflexion as a perceptron function when we use to NLL (not sure).

With naive bayes, we don't need a gradient descent and it count frequencies of the words in the corpus.

Also interesting when we compute the p(w_positive) we add +1 to the up and V to the down to smooth the results (in case of Zero exception).

Testing the Naive Model :

1) Predict function

We have everything we need to make our predictions (p) on some tweets.

$$p = logprior + \sum_{i}^{N}(loglikelihood_i)$$

The essential of this function would be to test if the word is in the likelihood dictionary. My code seems to have different results as the original notebook I had this observation during the implementation of the test_naives_bayes. The difference isn't huge don't worry it changed the prediction of 0.02 maximum.

2) Test function returning the accuracy

```
# Feel free to check the sentiment of your own tweet below
my_tweet = 'Je veux un bon stage à Londres'
naive_bayes_predict(my_tweet, logprior, loglikelihood)
```

If the prediction is >0, the predicted class is 1 and we append 1 to a list . Computing the error with the mean. It's a god way to check our sentence in your tweet. The accuracy is the known metrics and the most use.

Filter words by Ratio of positive to negative counts :

- Some words have more positive counts than others, and can be considered "more positive". Likewise, some words can be considered more negative than others.

- One way for us to define the level of positiveness or negativeness, without calculating the log likelihood, is to compare the positive to negative frequency of the word.

- Note that we can also use the log likelihood calculations to compare relative positivity or negativity of words.- We can calculate the ratio of positive to negative frequencies of a word.

- Once we're able to calculate these ratios, we can also filter a subset of words that have a minimum ratio of positivity / negativity or higher.

- Similarly, we can also filter a subset of words that have a maximum ratio of positivity / negativity or lower (words that are at least as negative, or even more negative than a given threshold).

```python
def get_ratio(freqs, word):
    '''
    Input:
        freqs: dictionary containing the words
        word: string to lookup

    Output: a dictionary with keys 'positive', 'negative', and 'ratio'.
        Example: {'positive': 10, 'negative': 20, 'ratio': 0.5}
    '''
    pos_neg_ratio = {'positive': 0, 'negative': 0, 'ratio': 0.0}

    # use lookup() to find positive counts for the word (denoted by the integer 1)
    lookup(freqs, word, 1)

    # use lookup() to find negative counts for the word (denoted by integer 0)
    lookup(freqs, word, 0)

    # calculate the ratio of positive to negative counts for the word
    pos_neg_ratio = (lookup(freqs, word, 1) + 1 ) /(lookup(freqs, word, 0) +1) #smoothing it

    return pos_neg_ratio
```

Note : for the positive_neg_ratio : I had one to the fraction (up dans down) to smooth it and not tend to 0 or create an error.

Conclusion found in the course.

Naïve Bayes Limitation

Assumption 1: words in a sentence are independent.

- That is normally wrong.
- The words that come before usually serve as a hint of the next one's meaning.
- We will see more intelligent models in the following classes.

Assumption 1: relies on the distribution of the training dataset.

- In the real word, we usually don't have balanced datasets.
- We end up having to synthetically balance the data.