Clustering tweets about Machine Learning using Self-Organizing Maps

Ignacio Dorado and Arnaud Le Doeuff

Abstract

In this document we report our proposal for the application of self organizing maps to cluster tweets of a specific topic: Machine Learning. A dataset of tweets matching this words was collected using Tweepy. Afterwards, those tweets were transformed into a more homogeneous format, tokenized and vectorized. Principal component analysis was performed to reduce the vectors dimensions. The data was clustered using Sompy, an efficient implementation for SOMs. Finally, some graphical visualization are displayed in order to provide a better understanding of the results.

Contents

1 Description of the problem **2**

2 Capture the tweets **2**

2.1 Set up the Twitter API 2

2.2 Search the API for our query 2

2.3 Creating a pandas DataFrame 3

3 Preprocessing **3**

3.1 Cleaning the strings 3

3.2 Creating the dictionary 3

3.3 Creating and reducing the vectorized representation 3

4 Clustering **4**

4.1 Dimensionality reduction (PCA) 4

4.2 Trainning a SOM 4

4.3 Visualize the clusters 6

4.4 WordCloud 7

5 Implementation **7**

6 Questions and Answers **7**

7 Conclusions **8**

1 Description of the problem

Micro-blogging platforms such as Twitter have emerged in recent years, creating a radically new mode of communication between people. Every day, 500 million users send more than 500 million tweets (6000 every second), on every possible topic. Interactions and communication in Twitter often reflect real-world events and dynamics, and important events like elections, disasters, concerts, and football games can have immediate and direct impact on the volume of tweets posted. Because of its real-time and global nature, many people use Twitter as a primary source of news content, in addition to sharing daily life, emotion and thoughts. Techniques like sentiment analysis are spreading widely across private companies and an increasingly number of studies are relying on this platform as a non-ending data provider. Yet this is not a one-way interaction, as well as data scientists work with Twitter data, Twitter talks about data scientists and machine learning. In this study both directions are present, our goal is to cluster those tweets talking about Machine Learning.

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group or cluster are more similar to each other than to those in other clusters. Aiming to perform this task, we rely on Self Organizing Maps, a type of artificial neural network (ANN) that produce a low-dimensional discretized representation of the input space of the training samples, called a map. SOMs do not require any label, is an unsupervised algorithm. Although they do require a discretization of the input. Since a preprocessing is necessary and highly important, we will divide our approach into 3 different steps.

- Capture the tweets
- Preprocessing
- Clustering

2 Capture the tweets

2.1 Set up the Twitter API

For collecting the tweets dataset, we use Tweepy [1]. Tweepy is python library that connect with the Twitter API and simplifies every interaction with it. It wrapps the Twitter API and hide complexity. This library has been widely used to successfully conduct several studies, like the project [2] made by Miguel Garcia Goni. Where he implement a bot who automatically follows anyone who follows you on tweeter.

In order to connect with the Twitter API, we need to use a Twitter Developer account and create an app. A extractor object is creating, using our account credentials for authentication. For privacy, those parameters (API Key, API Secret Key, Access Token, Access Secret Token) are defined into an external file.

2.2 Search the API for our query

The Twitter API establish some limits for tweet extraction. It only allow us to download a maximum of 3200 tweets every 15 minutes. We set up our API connection to wait when that maximum is reached and keep retrieving tweets when the limit counter restart.

The queried string was ”machine learning”. This query matchs every tweet containing those two words, either in the main text, as a hastag or in a link title. Using ”ML” as a query was also considered, but this was discarded because of the amount of noise for short acronyms.

A big number of the collected tweet set consisted on retweets (RTs), which are just reposted content that do not have any original text, hence they are not providing us any new information. All RTs were discarded: approximately 20% of the total. Eventually, 2302 non retweets tweets were collected.

2.3 Creating a pandas DataFrame

Pandas [3] is a free software library for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. We are interested in the DataFrame structure: two-dimensional, size-mutable, potentially heterogeneous tabular data.

All the tweets' full text were stored into a DataFrame, along with some other tweets features considered relevant. This data structure will notably simplify the data manipulation and also the file writing and reading.

3 Preprocessing

3.1 Cleaning the strings

Words are written in Twitter without any filter, and its grammar syntax and format is hard to predict and to work with. That is why our first step should be transforming the strings into a more homogeneous format. To accomplish that, we use Python regular expressions module. A regular expression or "regex" is a sequence of characters that define a search pattern. We proceed to describe those transformations.

- URLs were removed due to the lack of literal semantic meaning.
- Account tags were removed, account names in Twitter do not provide any information.
- Hastag symbols were removed, but the hastag string was treated as a word itself.
- Contractions, like "he's" or "don't" were split into it component words: "he" and "is", "do" and "not".
- Interrogations and exclamation marks were also split and treated as a separated symbol with its own semantic contribution.
- All punctuation symbols were removed.
- All other non alphabet symbols were removed, including numbers.
- White spaces were also stripped away.

3.2 Creating the dictionary

A vocabulary of 8037 unique words was extracted from the dataset and a dictionary matching each word to its amount of occurrences was created. 4589 words were observed to be present only once in the hole set of tweets. According to this information, we can conclude that most words in the vocabulary are not playing any significant role in the discovery of similarity between data.

For this task we used the text Tokenizer provided by Tensorflow [4], present in the Keras module.

3.3 Creating and reducing the vectorized representation

For the SOM algorithm to accept the input, we need to transform it into a numeric vector. Our choice was to transform each tweet into a vector of our vocabulary length, where each component stands for the amount of occurrences of the word with that same index.

This huge dictionary has some flaws, so this modifications were made:

- **Least frequent words:** as mentioned in the previous section, a big percentage of the words did not have many repetitions across the tweets, those words contribution to the extraction of common underlying patterns is really low. For this reason, and also to keep it simpler, we decide to work only with the 1000 most frequent words.
- **Most frequent words:** in any language, and specially in English, articles and preposition are so common that their frequency stand out over other words frequency. These short words usually lack of individual significance. For this reason, the components in the vectors referring to the 15 most common words were dropped.

Finally our set of tweets ended up as matrix containing 2302 vectors (one for each tweet) of 985 integers components (one for each word in our vocabulary).

4 Clustering

4.1 Dimensionality reduction (PCA)

For this high dimensional problem, a multivariate principal component analysis can be used to reduce the dimension before the output factors are becoming input in a clustering routine [5].

Principal component analysis (PCA) is a technique used for identification of a smaller number of uncorrelated variables, known as principal components, from a larger set of data. The technique is widely used to emphasize variation and capture strong patterns in a data set. The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes [6]. Table 1 gives a summary of the variance of the data for the principal components. In order to explain at least 60% of the data, it was required to use the first 120 components. This is not a great result, but it is still reducing much complexity and getting rid off almost 900 dimensions, having some loss of information as a pay off.

Other dimensionality reduction algorithms, such as Linear Discriminant Analysis (LDA) and Singular Value Decompostion (SVD) were also tested with similar or worse results.

Table 1: Cumulative variances explained

Component	Variance (%)	Cumulative (%)
1	4.45	4.45
2	2.79	7.24
3	2.33	9.57
4	2.04	11.61
5	1.65	13.27
-	-	-
20	0.76	28.54
-	-	-
40	0.47	40.40
-	-	-
60	0.35	48.50
-	-	-
80	0.27	54.69
-	-	-
100	0.22	59.58
-	-	-
120	0.18	63.51

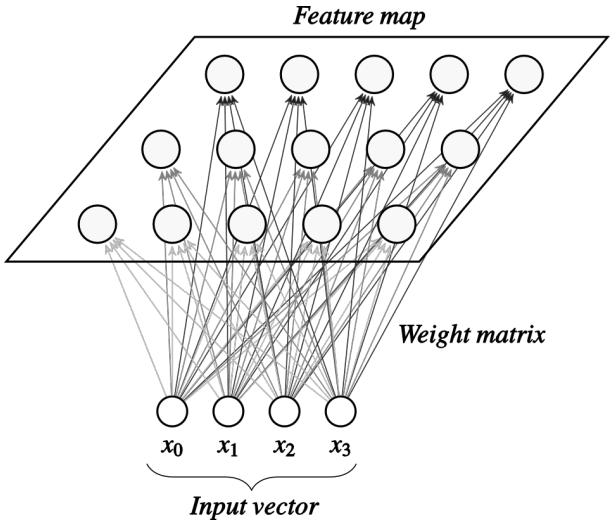
4.2 Trainning a SOM

A self-organizing map (SOM) is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction. Self-organizing maps differ from other artificial neural networks as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighborhood function to preserve the topological properties of the input space. Competitive learning is a ANN category in which neighboring cells compete in their activities by means of mutual lateral interactions, and develop adaptively into specific detectors of different signal patterns. Wheter a SOM can be considered as ”neural” algorithm or not has been a discussion [6].

In a SOM algorithm, the neurons are arranged in two layers: the input layer and the competition layer. The input layer is composed of N neurons, one for each input variable. The competition layer

is composed of a topological low-dimensional grid of neurons (usually 2-dimensional) geometrically ordered, as ilustrated in Figure 1. Every input is projected to the competition layer and the closer neuron to that projection is the winning neuron, usually referred as best matching unit (BMU). The BMU is the one neuron that learns from that experience, updating its weights and getting closer to the input. In contrast with "winner-takes-all" approaches, the BMU is not the only neuron learning, every other neuron in its topological neighborhood updates its weights in a different ratio that decays with the lateral distance from the BMU and is defined by the topological neighborhood function. Hence SOM algorithm can also be considered as cooperative algorithm, in addition to competitive.

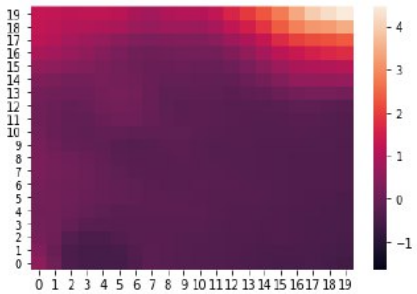
Figure 1: SOM neuron layers



For our trainning we use Sompy [7], a Python library for fast and parallel computation of self organzing maps. The chosen grid was consisting on a neuron matrix of 20 rows and 20 columns. The training only took a few seconds and it ended with a quantization error of 1.17. Quantization error is a measure of the average distance between the data points and the map nodes to which they are mapped, with smaller values indicating a better fit. Figure 2 shows the weights learned by each neuron in the grid for each component. It is also possible to observe how every neuron in the grid moved towards the centroids of the points, figure 3 illustrates that: blue points are the projections of every input, red points are the trained neurons and red lines represent neighboring relationships.

Figure 2: Heatmap for the weights of each component

(a) First component



(b) Second component

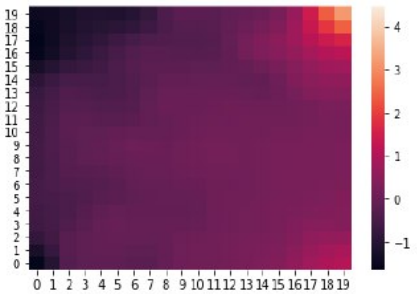
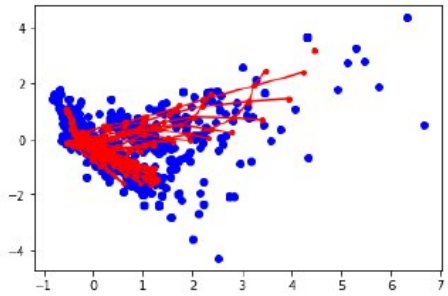


Figure 3: Projected inputs and adjusted neurons



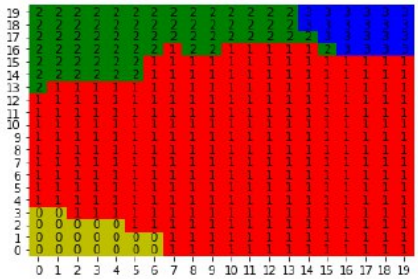
4.3 Visualize the clusters

The data was divided into 4 different clusters. In order to graphically understand the division, some plots were made:

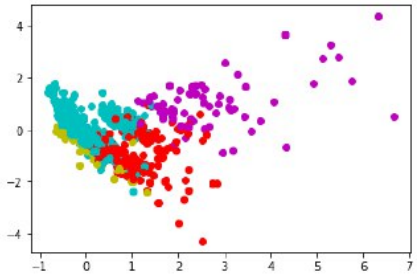
- Figure 4a represents the cluster associated to each neuron.
- Figure 4b shows the projected inputs, coloring each cluster with a different color.
- Figure 4c displays the unified distance matrix (U-Matrix) for the learned weights. The U-matrix is a representation of the Euclidean distance between the codebook vectors of neighboring neurons, depicted in a coolwarm scale image.

Figure 4: Cluster Visualizations

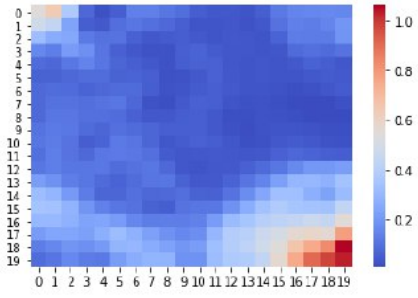
(a)



(b)



(c)



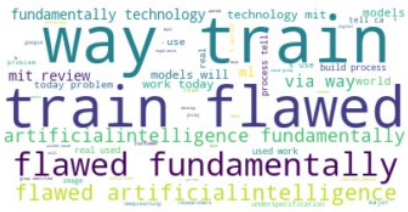
4.4 WordCloud

A Word Cloud is an image composed of words used in a particular text or subject, in which the size of each word indicates its frequency or importance. So, the more often a specific words appears in the text, the bigger and bolder it appears in the word cloud. Studies like [8] have used this kind of representations to depict their results.

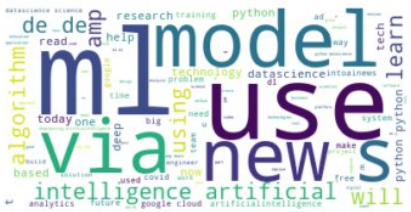
Figures 5 are the representations of our clusters.

Figure 5: Word Clouds

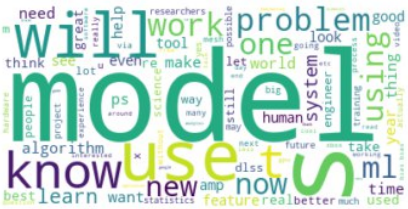
(a) Word Cloud for Cluster 0



(b) Word Cloud for Cluster 1



(c) Word Cloud for Cluster 2



(d) Word Cloud for Cluster 3

5 Implementation

All the project steps were implemented in Python. We use Tweepy for extracting the tweets; Pandas for reading and preprocessing the dataset; Tensorflow to create the word index; scikit-learn for the principal component analysis; Sompy for training the self organizing map and Seaborn to plot the heatmaps.

The dataset can be found in the file `tweets.csv` We illustrate how the implementation works in the Python notebook `ML_Tweet_Clustering_with_SOMs.ipynb`.

6 Questions and Answers

1. **What class of problems can be solved with the NN?**

   A SOM can be used to solve pattern recognition and clustering problems. It can also do a good work for dimensionality reduction. It does not requires any label, so it is meant to solve unsupervised problems.

2. **What is the network architecture?**

   The self-organizing map consist of two layers, the input layer and the output/competition layer. The parameters of the self-organizing map are the features and the values of the weight vectors. SOMs in these domains have generally been restricted to static sets of nodes connected in either a grid or hexagonal connectivity and planar or toroidal topologies.

3. **What is the rationale behind the conception of the NN?**

   The goal of learning in the self-organizing map is to cause different parts of the network to respond similarly to certain input patterns. This is partly motivated by how visual, auditory or other sensory information is handled in separate parts of the cerebral cortex in the human brain.

4. **How is inference implemented?**
   Because in the training phase weights of the whole neighborhood are moved in the same direction, similar items tend to excite adjacent neurons. Therefore, SOM forms a semantic map where similar samples are mapped close together and dissimilar ones apart.
5. **What are the learning methods used to learn the network?**
   A learning algorithm is used in order to calculate the associated weights for each competition layer neuron and a weight update is performed so that the winning neuron is approached to the input observation and also in its neighbors units. Is a competitive and cooperative model at the same time.

7 Conclusions

In our project we have applied PCA and Self Organizing Maps to a dataset of 2302 tweets, extracted from the Twitter. Our poor results with PCA shows the difficulty of natural language problems, as well as the importance of a good representation choice and a right preprocessing. Those steps has proven to be more of a key factor than the model training itself. Moreover, SOMs prove the value of visualization, being its graphical representations one of the it main advantages.

References

[1] Tweepy V 3.5 : An easy-to-use Python library for accessing the Twitter API. https://www.tweepy.org/.

[2] Miguel Garcia. How to make a twitter bot in python with tweepy. https://realpython.com/twitter-bot-python-tweepy/#:~:text=Tweepy%20is%20an%20open%20source,the%20Twitter%20API%20with%20Python.

[3] Pandas V 1.1.4 : An open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. https://pandas.pydata.org/pandas-docs/stable/index.html.

[4] Tensorflow V 2.3.0 : An Open Source Machine Learning Framework for Everyone. http://www.tensorflow.org.

[5] Sitanath Majumdar Manojit Chattopadhyay, Pranab K. Dan. Principal component analysis and self organizing map for visual clustering of machine-part cell formation in cellular manufacturing system : An Open Source Machine Learning Framework for Everyone. 2011.

[6] Takenori Kanai Suwardi Annas and Shuhei Koyama. Principal component analysis and self-organizing map for visualizing and classifying fire risks in forest regions . https://pdfs.semanticscholar.org/a0b0/f750a6c7ff72b582b49e33e58f62b151f0b9.pdf, 2007.

[7] V Moosavi, S Packmann, and I Vallés. Sompy: A python library for self organizing map (som), 2014. GitHub.[Online]. Available: https://github. com/sevamoo/SOMPY.

[8] Miguel Garcia. A Case Study in Text Mining: Interpreting Twitter Data From World Cup Tweets. https://arxiv.org/abs/1408.5427, 2014.