Sentiment Analysis via Classical ML and Non-Parametric Bayesian Techniques: a Tutorial Comparison

Claudio Battiloro
i3s dept.
La Sapienza University
Rome, Italy
battiloro.claudio@gmail.com

Federica Spoto
i3s dept.
La Sapienza University
Rome, Italy
fd.spoto@gmail.com

Supervisor: Brunero Liseo
MEMOTEF dept.
La Sapienza University
Rome, Italy
brunero.liseo@uniroma1.it

***Abstract*—In this paper we propose a Sentiment Analysis application to investigate and compare classical unsupervised and supervised ML techniques with Dirichlet Process Mixtures and Mixtures of finite Mixtures procedures.**

***Index Terms*—Sentiment Analysis, TfIdf, Word2Vec, KMeans, RF Classifier, MFM, DPM.**

I. Introduction

This work aims to present a consistent analysis and comparison between classical ML techniques and DPM and MFM (bayesian) procedures. In particular, we perform Sentiment Analysis on tweets plain text of U.S. Airline reviews to classify them as ”Negative” or ”Positive”.

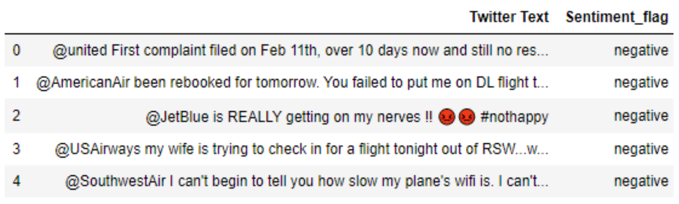
To achieve our goal, we first perform an embedding of the tweets via two different models (TfIdf and Word2Vec CBOW) and then work on them both in a supervised and an unsupervised fashion: in particular, we use as benchmark the Random Forest classifier accuracy and compare it with the accuracy reached by classical clustering techniques (K-Means and Spectral Clustering) and Bayesian clustering procedures based on Dirichlet and Finite Mixtures processes.

Moreover, various parameters and model configurations are used, enhancing, for each of them, the robust and weak points.

Our principal contribute is showing how the Bayesian approach is, also in this case, intrinsically more data-driven than the classical one, paying attention to what it means in the proposed task and why is it good choosing a procedure more than another.

II. The Data

The used dataset can be downloaded from [1] and it is composed by 6000 datapoints described through various features; for this task, we are interested only in tweets plain text and sentiment flags (the target). In order to have a clearer analysis not affected by different sample sizes, the target proportions are balanced. To have a better idea of the data structure, in [fig.1] is showed a small portion of the dataset.



	Twitter Text	Sentiment_flag
0	@united First complaint filed on Feb 11th, over 10 days now and still no res...	negative
1	@AmericanAir been rebooked for tomorrow. You failed to put me on DL flight t...	negative
2	@JetBlue is REALLY getting on my nerves !! 😡😡 #nothappy	negative
3	@USAirways my wife is trying to check in for a flight tonight out of RSW...w...	negative
4	@SouthwestAir I can't begin to tell you how slow my plane's wifi is. I can't...	negative

Fig. 1: Small portion of the dataset

A. Tools

The entire procedure is implemented using Python and Julia programming languages. The two languages are connected via the Jupyter Notebook suite [logo, fig. 2].

Fig. 2: Jupyter Notebook logo

B. Data Preprocessing

Once we extract the plain texts, we need to preprocess them by cleaning special characters, removing empty spaces and converting the entire corpus in lower case.

Moreover, to use word embedding, we need to decide how to tokenize the sentences [fig. 3]; it turns out that the best, intuitively, choice, is tokenizing by words, so converting each sentence in a vector of single words.

III. Word Embedding

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a mathematical embedding from a space with many dimensions (possibly infinite) per word to a continuous vector

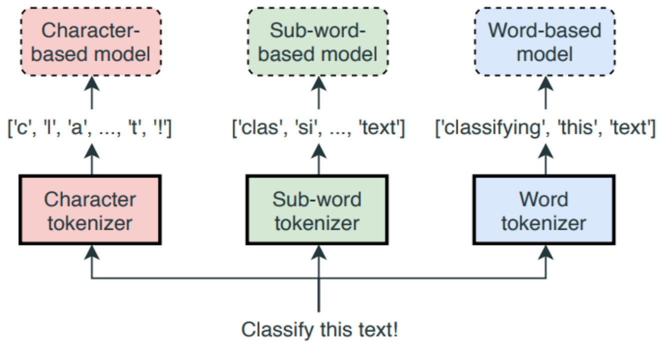


Fig. 3: Tokenization possibilities

space with a much lower dimension. Formally, we look for an embedding mapping $e(\cdot)$ from the tweets complete dictionary set $\mathcal{C}$ to a (priori) fixed dimension $\mathbb{R}^D$.

A. Models Principles

Nowadays, this mappings are mainly focused on two (opposite) principles:

1) BagOfWords models: the BagOfWords philosophy is based on a simplifying text representation. In this kind of models, a text is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. In this work, we use one of this models called TfIdf, short for term frequency–inverse document frequency; it is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.
The tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.
   - TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

$$\mathbf{tf_{i,j}} = \frac{\mathbf{n_{i,j}}}{|\mathbf{d_j}|}$$

   where $n_{i,j}$ is the number of occurrences of the term i in the document j, and the denominator $|d_j|$ is the dimension of the document j expressed as the number of words it contains.
   - IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$\mathbf{idf_i} = \log \frac{|\mathbf{D}|}{|\{\mathbf{d} : \mathbf{i} \in \mathbf{d}\}|}$$

   where $|D|$ is the number of the documents in the corpus, and the denominator is the number of documents that contain the term i.

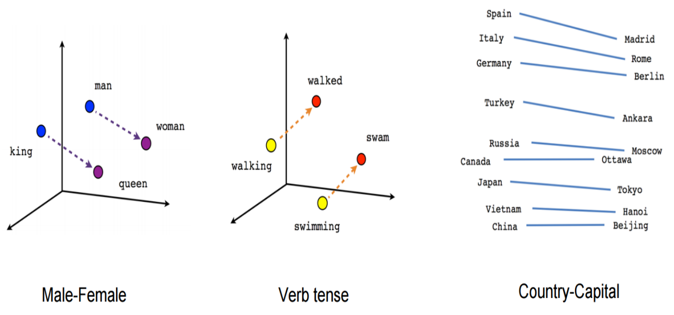


Fig. 4: CBOW embedding examples

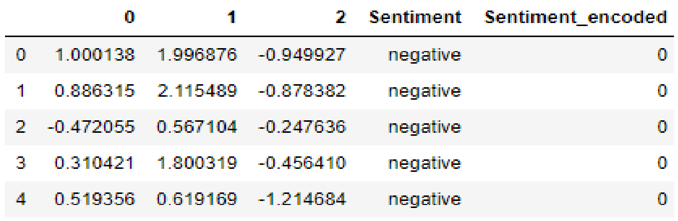
2) Word2Vec models: these models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words. Word2vec produces a vector space, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space. In this work, we use one of this models called CBOW. In [fig. 4] we show an example of how context-related words are mapped in similar locations. Due to its difficulty, a deeper review can be found in [2].

B. Embedding set up

In our work, we performed a GridSearch to choose the best models hyperparameters configuration. We use:

- TfIdf: this method builds the vocabulary with the top 2000 words ordered by term frequency, removing the english stopwords, ignoring terms that have a document frequency strictly higher than 0.6 or appear in less than 3 documents.

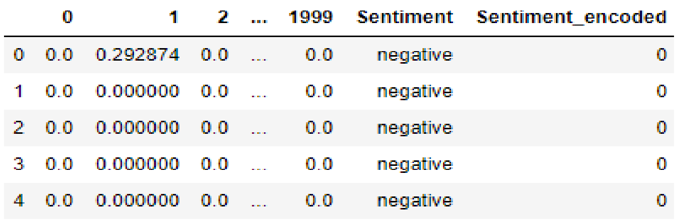
- Word2Vec: this model intrinsically allows us to reduce the embedding dimension. In particular, the best possible representation for our (relatively small) tweets corpus is a (only) 3D vector per each word. This is a powerful result and enhances the importance of embedding also the words context in the mapping procedure. The final 3D embedding of the tweets is obtained as the average of the values of its words per each component.



	0	1	2	Sentiment	Sentiment_encoded
0	1.000138	1.996876	-0.949927	negative	0
1	0.886315	2.115489	-0.878382	negative	0
2	-0.472055	0.567104	-0.247636	negative	0
3	0.310421	1.800319	-0.456410	negative	0
4	0.519356	0.619169	-1.214684	negative	0

Fig. 5: Word2Vec embedding

In [fig. 5] and [fig. 6] some entries of the embedded dataset are shown. As the reader can see, the Word2Vec embedding is drastically more efficient in term of features dimension and information compression.



	0	1	2	...	1999	Sentiment	Sentiment_encoded
0	0.0	0.292874	0.0	...	0.0	negative	0
1	0.0	0.000000	0.0	...	0.0	negative	0
2	0.0	0.000000	0.0	...	0.0	negative	0
3	0.0	0.000000	0.0	...	0.0	negative	0
4	0.0	0.000000	0.0	...	0.0	negative	0

Fig. 6: TfIdf embedding

IV. Classical Machine Learning Approach Results

Once we get the embedded data, it's possible to work on them as usual in the most of ML procedures. First of all, we are interested in giving an accuracy benchmark through a supervised approach, in particular we trained a Random Forest classifier on the data. Then, we compared its results with a pair of unsupervised algorithm (clustering-based) like K-Means and Spectral Clustering. In the following sections the obtained results are presented together with algorithms configurations.

A. *Random Forest Classifier*

A brief review of the method can be found in [3]. The algorithm has the following parameters:
- $n\_estimators = 100$, i.e. the number of trees in the forest
- $random\_state = 0$, i.e. the seed used to control the randomness of the algorithm.

B. *K-Means*

A brief review of the method can be found in [4]. The algorithm has the following parameters:
- $n\_clusters = 2$, i.e. the number of clusters generated
- $random\_state = 0$, i.e. the seed used to control the randomness in choosing the initial centroids.

C. *Spectral Clustering*

A brief review of the method can be found in [5]. The algorithm has the following parameters:
- $n\_clusters = 2$, i.e. the number of clusters generated
- $random\_state = 0$, i.e. the seed used to generate a pseudo random number used for the initialization of the eigen vectors decomposition.

The results of these algorithms are presented in [table 1] for both the proposed embedding methods. As we can see, for what concerns the accuracy benchmark given by the Random Forest Classifier, it is higher when evaluating the TfIdf sketch, so when the algorithm receives an input with more even if sparser features; while in the unsupervised framework the compactness of the Word2Vec representation gives better results. Here, moreover, the classification obtained with K-Means seems to be more accurate; the low performances of Spectral Clustering are probably due to the way the algorithm builds the associated graph, in particular the embedding mapping seems to be not completely encoded and makes the procedure converging substantially to one big cluster.

	Random Forest Classifier	K-Means	Spectral Clustering
TfIdf	0.916	0.622	0.528
Word2Vec	0.855	0.748	0.630

Table 1: Results for the two embeddings

D. *On Clusters Number*

Until now, the presented methods need the number of cluster to be explicitly specified as an hyperparameter. The reader could ask which is the ratio behind this choice; unfortunately, there are no theoretical guaranteed methods to choose the number of clusters. For this application, of course we know that we're looking for two clusters ("Negative" and "Positive" reviews) but, in general, some nice heuristics can be used; the most famous one is the so called "Elbow Method": this method looks at the percentage of variance explained as a function of the number of clusters. One should choose a number of clusters so that adding another cluster doesn't give much better modeling of the data. More precisely, if one plots the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the "elbow criterion". Notice that the "elbow" cannot always be unambiguously identified. In [fig. 7] (we use the reconstruction error as target function, so it's $1-$ the explained variance and the graph is flipped) we can appreciate how the "suggested" number of clusters for K-Means is again 2 using the proposed heuristic with the Word2Vec embedding. Here, using this method with the TfIdf embedding would not add information on the suggested number of clusters, given the negative affect of the high number of features and their sparsity on the analysis of the reconstruction error.

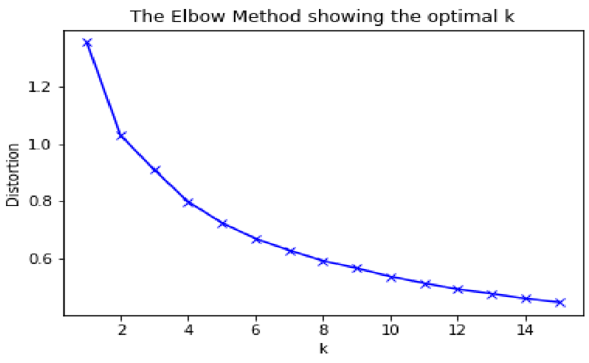


Fig. 7: Elbow Method for K-Means

V. The Bayesian Approach: MFM and DPM

In this section we propose an alternative clustering (but not only) framework based on Non-Parametric Bayesian estimation theory of Mixtures Finite Models and Dirichlet Process Mixture. The proposed procedures follows from [6].

We assume the reader is familiar with Bayesian approach and DPM and MFM models. Anyway, in the following sections, we review some of the fundamental concepts behind these models.

A. Dirichlet Processes and Stick-breaking Representation: a Constructive Definition

It is known that draws from a DP are composed of a weighted sum of point masses. In [7] this is made precise by providing a constructive definition of the DP as such, called the stick-breaking construction. This construction is also significantly more straightforward and general than previous proofs of the existence of DPs. It is simply given as follows:

$$\begin{gathered}\beta_K \sim Beta(1,\alpha)\ i.i.d \\ \theta_K^* \sim H\ i.i.d \\ \pi_k = \beta_k \prod_{l=1}^{K-1}(1-\beta_l) \\ G = \sum_{k=1}^{\infty}\pi_k\delta_{\theta_k^*}\end{gathered} \tag{1}$$

Then $G \sim DP(\alpha, H)$. Because of its simplicity, the stick-breaking construction has lead to a variety of extensions as well as novel inference techniques for the Dirichlet Process.

B. Kernel Smoothing

For the most of the applications, it's drastically limiting having a tool that produces only discrete distribution. We would like to essentially have densities without relying on parametric models. The solution is to smooth out draws from the DP with a kernel. Let $G \sim DP(\alpha, H)$ and let $f(x|\theta)$ be a family of densities (kernels) indexed by θ. We use the following as our nonparametric density of x:

$$p(x) = \int_\Theta f(x|\theta)dG \tag{2}$$

C. Mixture Models

The most common application of the Dirichlet process is in clustering data using mixture models. Here the nonparametric nature of the Dirichlet process translates to mixture models with a countably infinite number of components. We model a set of observations $x_1, ..., x_n$ using a set of latent parameters $\theta_1, ..., \theta_n$. Each θ_i is drawn independently and identically from G, while each x_i has distribution $F(\theta_i)$ parametrized by θ_i:

$$\begin{gathered}x_i|\theta_i \sim F(\theta_i) \\ \theta_i|G \sim G \\ G|\alpha, H \sim DP(\alpha, H)\end{gathered} \tag{3}$$

Because G is discrete, multiple θ_i's can take on the same value simultaneously, and the above model can be seen as a mixture model, where x_i's with the same value of θ_i belong to the same cluster.

This statement can be made explicit by observing that, if the data distributions $F(\theta)$ are smooth with densities given by $f(x|\theta)$, then holds:

$$p(x) = \int_\Theta f(x|\theta)dG = \sum_{k=1}^{\infty}\pi_k f(x|\theta_k) \tag{4}$$

Where π_k are defined as in (1). It's true that the DP mixture model is an infinite mixture model—a mixture model but, because the π_k's decrease exponentially quickly, only a small number of clusters will be used to model the data a priori (in fact the expected number of components used a priori is logarithmic in the number of observations).

D. Limits and Inconsistency for the number of component in DPM

The presented model is widely used and many applications show its effectiveness. However, a huge drawback can be found with a deeper analysis. As explained in [8], it is well-known that Dirichlet process mixtures (DPMs) are consistent for the density — that is, given data from a sufficiently regular density p_0 the posterior converges to the point mass at p_0. However, it is easy to see that this does not necessarily imply consistency for the number of components, since for example, a good estimate of the density might include superfluous components having vanishingly small weight. So it is important to understand the behavior of the number of components posterior when the data comes from a finite mixture — in particular, does it concentrate at the true number of components?

Unfortunately the answer is no. Again in [8], a simple DPM model is presented and its shown how the aforementioned inconsistency turns out. In particular, if we consider the model in (3), it can be shown that the a priori probability of obtaining a certain induced partition A by n data with t components (clusters, $|A| = |\{A_1, .., A_t\}| = t$) is equal to (derived from the CRP [9] formulation) :

$$p(\mathcal{A} = A, T = t|n) = \frac{\alpha^t}{\alpha^{(n)} t!} \prod_{i=1}^{t} (|A_i| - 1)! \tag{5}$$

The inconsistency can be proved by putting us in an extreme case. Suppose we sample n data from one single $f(\cdot|\theta_i)$, it can be shown that:

$$\lim_{n\to\infty} p(T = 1|X_{1:n} = x_{1:n}) < 1 \tag{6}$$

So the process is not able to recognize one single cluster and still prefers including some extracomponent in density estimation.

A natural solution is the following: if the number of components is unknown, put a prior on the number of components. Under certain mild conditions, the posterior on the density in this case has been shown to concentrate at the true density and true components number at the minimax-optimal rate, up to a logarithmic factor, for any sufficiently regular true density. This kind of models are named as Mixture of Finite Mixtures. In the following sections we investigate them and apply them to our task.

E. Mixture of Finite Mixtures

The MFM model is clearly another kind of mixture-based model. Thanks to the aforementioned theoretical guarantee, it is a powerful tools to perform non-parametric inference. Moreover, it turns out that many MFM features are strictly related or similar,in their formulation, to the DPM ones and this allowed developing tools (i.e. samplers) based on previous acknowledgements about DPM.

In this section, we wanna introduce the model and emphasize on which aspects it's similar or not to DPM. The model is the following one:

$$\begin{gathered} x_i|\theta_{1:K}, Z_{1:n} \sim F(\theta_{z_i}) \ i.i.d. \\ \theta_1, ..., \theta_k|H, K = k \sim H \ i.i.d. \\ Z_1, ..., Z_n|\pi \sim \pi \ i.i.d. \\ \pi = (\pi_1, ..., \pi_k)|K = k \sim Dir(\gamma) \\ K \sim p_k \end{gathered} \tag{7}$$

Where p_k is a p.m.f. on $\mathbb{N}$ and H is a prior or "base measure" on Θ, and $\{f_\theta : \theta \in \Theta\}$ is a family of probability densities with respect to a sigma-finite measure ζ on $X \subset \mathbb{R}^D$. We invite the reader to go deeper into technical conditions on model component; a very nice explanation can be found in [6] (indeed we follow it in this work).

Notice that prior information about the relative sizes of the weights $\pi_1, ..., \pi_k$ can be introduced through γ —roughly speaking, small γ favors lower entropy π's, while large γ favors higher entropy π's. Also note that sometimes it is useful to put a hyperprior on the parameters of H.

As we done it for DPM in (5), it's possible to show that, for MFM (keeping the same notation):

$$p(\mathcal{A} = A, T = t|n) = V_n(t) \prod_{i=1}^{t} \gamma^{(|A_i|)} \tag{8}$$

Where:

$$V_n(t) = \sum_{k=1}^{\infty} \frac{k_{(t)}}{(\gamma k)^{(n)}} p_K(k) \tag{9}$$

The fact that we can, both for MFM and DPM, introduce an explicit distribution over the induced partitions, allows as to unify the statement of the models by simply taking it into account:

$$\begin{gathered} A, t \sim p(A, t) \\ \phi_1, ..., \phi_t|A, t \sim H \ i.i.d \\ X_j|A, t, \phi_j \sim f_{\phi_j} \ forj = 1, ..., t \end{gathered} \tag{10}$$

The previous model is a DPM if $p(A, t)$ is as in (5), instead it is a MFM if $p(A, t)$ is as in (8).

This representation is particularly useful for doing inference, since one does not have to deal with component labels or empty components.

F. Realizations of an MFM

It can be shown that all the realizations of (7) have the form of a finite mixture model so that:

$$p(x) = \sum_{k=1}^{K} \pi_k f(x|\theta_k) \tag{11}$$

So the model can be equivalently expressed, as in DPM, with Random Discrete Measure notation as:

$$\begin{gathered} x_i|\theta_i \sim F(\theta_i) \\ \theta_i|G \sim G \\ G|\gamma, H, p_K \sim MP(\gamma, H, p_K) \end{gathered} \tag{12}$$

Just to close the circle, also for MFM (as DPM) it is possible to derive the process properties through a Restaurant Scheme. DPM and MFM are, indeed, two examples of a more general class of processes based on the urn scheme.

In [table 2] is shown a partial comparison between MFM and DPM. Of course, many deeper considerations can be done, again [6] is a good reference.

DPM	MFM
$\beta_K \sim Beta(1,\alpha)\ i.i.d$ $\theta^*_K \sim H\ i.i.d$ $\pi_k = \beta_k \prod_{l=1}^{K-1}(1-\beta_l)$ $G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta^*_k}$ then $G \sim DP(\alpha, H)$	$K \sim p_k$ $\pi = (\pi_1, ..., \pi_k) \vert K = k \sim Dir(\gamma)$ $Z_1, ..., Z_n \vert \pi \sim \pi\ i.i.d.$ $\theta_1, ..., \theta_k \vert H, K = k \sim H\ i.i.d.$ $x_i \vert \theta_{1:K}, Z_{1:n} \sim F(\theta_{z_i})\ i.i.d.$
Partition Distribution $p_D(A, t\vert n) = \frac{\alpha^t}{\alpha^{(n)} t!} \prod_{i=1}^{t}(\vert A_i\vert - 1)!$ if there is a prior on α $p_D(A, t\vert n) = V_n(t) \prod_{i=1}^{t}(\vert A_i\vert - 1)!$ where $V_n(t) = \mathbb{E}_\alpha \left[\frac{\alpha^t}{\alpha^{(n)}}\right]$	**Partition Distribution** $p_M(A, t\vert n) = V_n(t) \prod_{i=1}^{t} \gamma^{(\vert A_i\vert)}$ where $V_n(t) = \sum_{k=1}^{\infty} \frac{k_{(t)}}{(\gamma k)^{(n)}} p_K(k)$
Equivalent partition based model $A, t \sim p_D(A, t)$ $\phi_1, ..., \phi_t \vert A, t \sim H\ i.i.d$ $X_j \vert A, t, \phi_j \sim f_{\phi_j}\ for j = 1, ..., t$	**Equivalent partition based model** $A, t \sim p_M(A, t)$ $\phi_1, ..., \phi_t \vert A, t \sim H\ i.i.d$ $X_j \vert A, t, \phi_j \sim f_{\phi_j}\ for j = 1, ..., t$
Process based model $G \vert \alpha, H \sim DP(\alpha, H)$ $\theta_i \vert G \sim G$ $x_i \vert \theta_i \sim F(\theta_i)$	**Process based model** $G \vert \gamma, H, p_K \sim MP(\gamma, H, p_K)$ $\theta_i \vert G \sim G$ $x_i \vert \theta_i \sim F(\theta_i)$

Table 2: Comparison between DPM and MFM

VI. MFM Approach Results

The previous emphasized clustering-property intrinsically encapsulated in MFM and DPM allows us to use them for our Sentiment Analysis task. We decided to use only MFM models because we're sure the number of cluster has to be finite and small, and we care about the consistency of the results. The entire implemented procedure is performed using the ad-hoc code in [10]. There are mainly three problems to face in this scenario:

- The clustering procedure is based on MCMC sampling of posterior distributions (how does the induced partition distribution change in light of data?). So we have to take it into account by choosing dimension-reduced data. The issue is clearly and easily solvable using the Word2Vec embedding, that allow us to use small 3-variate data points.

- It's crucial to design suitable MCMC sampling algorithms for MFM. For this task we noticed that the most performing one is the split-merge sampler of Jain and Neal [11], as explained in [6].

- How the results have to be interpreted? With the Bayesian approach, the entire model (except priors, that we choose uninformative to let the mechanism be completely transparent) is Data-Driven, so the clusters will not necessarily have the meaning we're looking for. We'll see how this apparent drawback can be successfully interpreted for our purposes.

A. Used Model for Sentiment Analysis

Thanks to the Word2Vec fundamental principles which tends to group words based on their context without (explicit) biases, it's legit to suppose that the data will follow a symmetric distributions family mixture. So we choose to use MVGaussian mixtures with diagonal covariance matrices, i.e., the dimensions are independent univariate Gaussians, and we place independent conjugate priors on each dimension; then we put an improper uniform distribution on the number of cluster K. The model can be written as:

$$\begin{gathered} \lambda \sim Gamma(a, b) \text{ where } \lambda \in \mathbb{R}^3 \\ \mu \vert \lambda \sim N(0, diag((c\lambda)^{-1})) \text{ where } \mu \in \mathbb{R}^3 \\ x \sim N\mu, \lambda^{-1}) \\ K \sim Uniform \end{gathered} \tag{13}$$

Before the implementation we normalized our data in order to be zero mean and unit variance in each of the three dimensions; so we choose $a = 1$, $b = 1$, and $c = 1$.

To do inference, we run the split-merge sampler of Jain and Neal [11] for conjugate priors with 2,000 samples.

B. Clustering Results

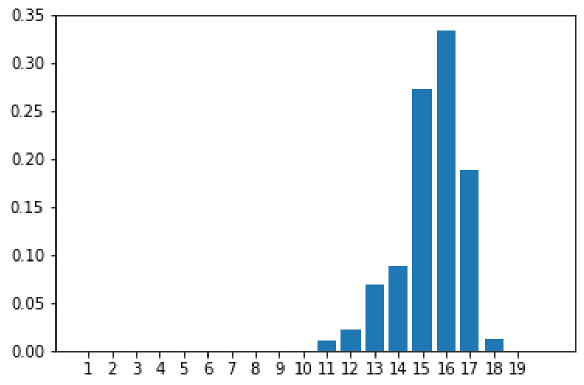


Fig. 8: Posterior distribution on the number of clusters

In [fig. 8] we plot the posterior distribution on the number of clusters: as we can see, the methods strongly converges to having 16 clusters! How can we read this result? Of course, at least at this point, it's not possible to compare the method to our RF benchmark.

The insight for the interpretation comes from [fig. 9] where we plot the data point indices against the inferred clusters; the first 3000 indices are the negative ones, the last 3000 are the positive ones. As we can see, it's true that MFM lead to an higher number of clusters, but it seems to introduce exclusive clusters per each review category (for example, the clusters $\{1, 2, 14, 15, 16\}$ contains only the positive reviews), admitting at the same time the presence of some "blurred" sentiment. This allow us to give a consistent possible interpretation of what the model outputs: the complete data-driven procedures identifies more than 2 sentiments, collecting a sort of sentiments gradient. There are effectively two macro-clusters, and we can extract them with a simple and intuitive procedure, explained in [alg. 1]. The main idea is to associate each MFM cluster, representing it as a single mean point, to one of the two big clusters running K-Means on the obtained 16 points with the number of clusters $k = 2$.

As we can see in [fig. 10], the method identifies the same macro-clusters we visually got before in [fig. 9]. Once we

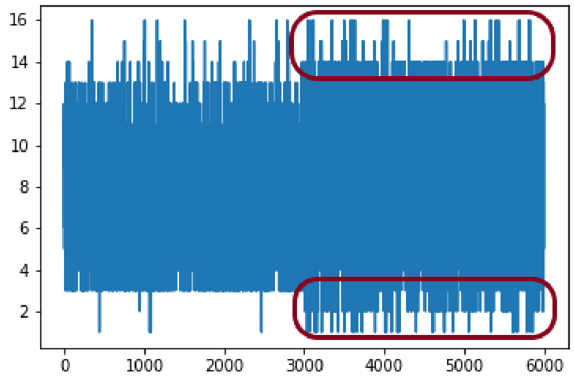


Fig. 9: Data index versus MFM clusters

```
Algorithm 1: Clustering on the MFM labels
  Input: $\mathcal{C} = \{C_1, C_2, ..., C_{16}\}$ MFM partition
1 Find the centroids for all the initial clusters:
      $$x_i^c = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_j, \quad i = 1, ..., 16, \; x_j \in \mathbb{R}^3$$
2 Run K-Means on $\{x_j^c\}_{j=1}^{16}$ with $k = 2$
  Output: $\mathcal{C}_M = \{C_{M_1}, C_{M_2}\}$ macro partition
```

extract the 2 clusters, we can compare them with the true target, and we obtain an accuracy of 0.68, similar to classical methods, but with an entirely data-driven approach and with the possibility of gaining meaningful extra-information.

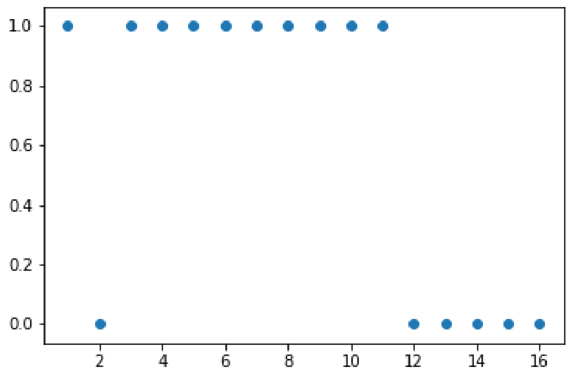


Fig. 10: MFM Clusters vs Associated Macro-Cluster

VII. Comments and Conclusion

In this paper we evaluate two philosophically different approaches. In classical ML the prior information is encoded in the model by explicitly constraining the related optimization problems (the probabilistic interpretation of this approach is given mainly by Gaussian Process literature); on the other side, with the Bayesian approach is possible to quantify more precisely the uncertainty about prior knowledge in a rigorous probabilistic framework and this is drastically useful when we look for a statistical complete characterization of the information contained in the data. Indeed, in this work we gain notable results by using a non-parametric approach and uninformative prior information, letting the data speak for themselves. Actually we show how both philosophies reach almost the same results (clearly in the unsupervised framework). Moreover, we appreciated various aspects per each approach, and from these we can understand that the capacity of manage both of them is fundamental in choosing the right strategy for tackling a certain problem.

References

[1] Kaggle, ”Twitter US Airline Sentiment”.
[2] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, September 2013.
[3] L. Breiman, Random Forests, January 2001.
[4] D. Arthur, S. Vassilvitskii, k-means++: The Advantages of Careful Seeding, January 2007.
[5] U. von Luxburg, A Tutorial on Spectral Clustering, 2007.
[6] J. W. Miller, M. T. Harrison, Mixture Models With a Prior on the Number of Components, 2017.
[7] J. Sethuraman, A constructive definition of Dirichlet priors, 1994.
[8] J. W. Miller, M. T. Harrison, A simple example of Dirichlet process mixture inconsistency for the number of components, January 2013.
[9] P. Frazier, I. Mukherjee, Bayesian nonparametrics, September 2007.
[10] Github, BayesianMixtures.jl.
[11] S. Jain, R. M. Neal, A Split-Merge Markov Chain Monte Carlo Procedure for the Dirichlet Process Mixture Model, March 2004.