Deep Clustering of Event Sequences

Vlad Zhuzhel [1] Evgeny Lagutin [1] Anna Rudenko [1] Liliya Mironova [1]

Abstract

Clustering temporal point processes is a challenging task due to the random nature of the point processes. There are a lot of methods for supervised and unsupervised classification of classical (synchronous) time-series. For asynchronous time-series or temporal point processes, there are almost no methods that exist right now. Dirichlet Hawkes Mixture Model allows to do unsupervised classification, however, this method is parametric. In this work, the novel Cohortney method has been taken and some different neural approaches have been used for clustering. That allowed to do non-parametric learning and achieve good results on both synthetic and real-data datasets. [1]

1. Introduction

In the world, there are a variety of processes that can be considered temporal point processes, e.g. actions on tweets, watching one of the channels with TV, activity on websites, accidents, the electronic health records in a hospital and many others. Indeed, all these processes can correspond to several clusters, for example, the website users can be loyal or non-loyal. In many cases, we don't have labels and need to perform learning in an unsupervised manner.

There are many well-established methods for synchronous data clustering with discrete time-lags of observations (Liao, 2005). However, there are few methods for clustering temporal point processes. The reason is that this data is in the continuous time domain, and it is hard to find the representation of this data. Also, the random nature of these processes makes it hard to learn clusters on this data.

The only existing solution in our notion that solves this problem is a Dirichlet Mixture Model of Hawkes Processes(Xu & Zha, 2017). The main idea of this method is to consider the Hawkes process parameters to be random variables and then learn the parameters in an expectation-maximization manner. The main drawback of this method, in our opinion, is that it has a parametric assumption and that these processes are Hawkes processes.

In this work, we propose non-parametric methods for temporal point processes clustering. To do so, we propose to use Cohortney partitions and then use different deep learning clustering techniques to obtain these clusters. It has been shown that this method allows us to obtain good clusters even if we consider small partitions and, in some cases, it can outperform DMHP algorithm.

The main contributions of this report are as follows:

- We successfully adopted a state-of-the-art unsupervised method proposed for image classification to event sequences classification task.
- We implemented different architectures for event sequences classification.
- We compared the performance of different architectures.

2. Preliminaries

Asynchronous sequential data is a kind of information that one may consider to analyze. The conventional example is an IPTV system. A common model covering most of the sequential data examples is called the event sequences model. Each sequence consists of events of different types in continuous-time. Clustering of these event sequences may be beneficial to the data owner: e.g. TV company is interested in clustering IPTV users to suggest ads to them according to their preferences.

A temporal point process is a random process whose realization consists of an event sequence $\{(t_i, c_i)\}_{i=1}^{M}$ with timestamps $t_i \in [0, T]$ and event types $c_i \in \mathcal{C} = \{1, \ldots, C\}$. It can be equivalently represented as C counting processes $\{N_c(t)\}_{c=1}^{C}$, where $N_c(t)$ is the number of type-c events occurring at or before time t. A way to characterize point processes is via the intensity function $\lambda_c(t) = \mathbb{E}\left[dN_c(t) \mid \mathcal{H}_t^{\mathcal{C}}\right]/dt$, where $\dot{\mathcal{H}}_t^{\mathcal{C}} = \{(t_i, c_i) \mid t_i < t, c_i \in \mathcal{C}\}$ collects historical events of all types before time t.

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Rodrigo Rivera-Castro <rodrigo.riveracastro@skoltech.ru>.

Final Projects of the Machine Learning 2020 Course, Skoltech, Moscow, Russian Federation, 2020.

[1]github.com/rodrigorivera/mds20_cohortney

One of the simplest types of point processes is Poisson process. In this case every event type has intensity λ_c and the probability of the event to occur is $P(t_i^c \in [t, t+dt)) = \lambda_d t$. Also one can consider homogeneous Poisson process, when $\lambda \neq \text{const}$ and $\lambda(t) \geq 0$.

However, in this case, the appearance of the new event doesn't depend on history. One of the generalizations is the Hawkes process.

In Hawkes process we assume that the history of the events can influence the intensity function:

$$\lambda_c(t) = \mu_c + \sum_{t_i<t} \phi_{cc_i}(t - t_i),\ \mu_c \geq 0,\ \phi_{cc'}(s) \geq 0 \quad (1)$$

μ_c is a base intensity, $\phi_{cc'}(s)$ are so called impact functions or kernels.

Let's assume that one has K type of Hawkes processes and a set of event sequences $\mathbf{S} = \{\mathbf{s}_n\}_{n=1}^N$, $\mathbf{s}_n = \{(t_i, c_i)\}_{i=1}^{M_n}$, $c_i \in \mathcal{C} = \{1, \ldots, C\}$ and $t_i \in [0, T_n]$. Each event sequence corresponds to one of K types of Hawkes processes with its intensity function.

One needs to perform clustering of the given set of event sequences.

3. Related work

As has been said there are few methods for the unsupervised classification of temporal point processes. In this work, we have used DMHP as a baseline and compared the performance with the performance of this method.

3.1. Dirichlet Mixture Model of Hawkes Processes

There are not many works doing clustering of event sequences and it seems to be no fully-fledged comparison of existing approaches in this direction. Therefore it is not very clear if there is a real SOTA approach. But still, as our main baseline, we take the method from (Xu & Zha, 2017). The authors represent event sequences as Hawkes Processes' realizations. The model looks as follows:

$$\lambda_{c_i}^k(t_i) = \mu_{c_i}^k + \sum_{t_j<t,d} a_{c_i,c_j,d}^k g_d(t - t_j) \quad (2)$$

, where g_d - basis functions that are Gaussians in the original paper, μ_c^k, $a_{c,c',d}^k$ - parameters of the model, $k \in 1..K$ - cluster indexes.

In the proposed mixture model, the probability of the appearance of an event sequence s is

$$p(\boldsymbol{s}; \boldsymbol{\Theta}) = \sum_k \pi^k \mathrm{HP}\left(\boldsymbol{s} \mid \boldsymbol{\mu}^k, \boldsymbol{A}^k\right), \quad (3)$$

$$\mathrm{HP}\left(\boldsymbol{s} \mid \boldsymbol{\mu}^k, \boldsymbol{A}^k\right) = \prod_i \lambda_{c_i}^k(t_i) \exp\left(-\sum_c \int_0^T \lambda_c^k(s) ds\right) \quad (4)$$

Here HP - PDF of Hawkes Process, π^k's are the probabilities of clusters. They propose learning parameters of each cluster with the nested EM algorithm. For that purpose, they introduce π^k - probabilities of each cluster. Therefore the likelihood of each sequence is a mixture of Hawkes processes. An important assumption the authors make in their paper is that μ^k, $\mathbf{A}^k$ are random variables and π is a random vector that obeys Dirichlet distribution. Their motivation for inducing priors for μ and A is that they have to preserve some physically-meaningful constraints. It is not absolutely clear why do authors need clusters to have Dirichlet prior or any prior at all. As a consequence, they use variational Bayesian inference to learn both the parameters of each model and their priors. Not going deep into details the joint distribution is factorized as

$$p(S, Z, \pi, \mu, \mathbf{A}) = p(S|Z, \mu, \mathbf{A})p(Z|\pi)p(\pi)p(\mu)p(\mathbf{A}) \quad (5)$$

$$p(S|Z, \mu, \mathbf{A}) = \prod_{n,k} \mathrm{HP}(s_n|\mathbf{A}^k \mu^k)^{z_{nk}} \quad (6)$$

$$p(Z|\pi) = \prod_{n,k} (\pi^k)^{z_{nk}} \quad (7)$$

Here S are N available event sequences, $Z \in \{0, 1\}^{N \times K}$ - matrix of labels.

To learn the above model authors propose using EM algorithm applied to a variational distribution $q(Z, \mu, A, \pi) = q(Z)q(\pi, \mu, A) = q(Z)q(\pi)\prod_k(\mu^k)q(A^k)$. At each E-step of the loop, they evaluate the expected value of $\log q^*(Z)$ and at M-step they evaluate the parameters μ, A maximizing the term $q(\pi, \mu, A)$ and update their priors, assuming μ, A to be equal to their expectation. It worth saying that inside the M-step it is needed to conduct an optimization procedure to obtain optimal μ and A and therefore authors provide an explicit iterative procedure that is inner EM loop at each M-step of the outer loop. Besides the main framework for clustering, the authors propose a way to update the number of clusters during the algorithm running. Moreover, they prove a theorem that under mild assumptions their mixture model is locally identifiable, meaning that for each set of parameters and cluster probabilities there is an open neighborhood containing no other set, such that likelihood given those two sets match for any possible sequence.

The authors provide a comparison of their method to several other approaches that may be described in a similar way: they choose a way to learn representations of the event sequences and apply the Dirichlet Process Gaussian Mixture Model (Krueger et al., 2018) to perform clustering. According to the paper (Xu & Zha, 2017), the solution proposed by the authors allows to get higher consistency of clustering on the unlabeled real-world data (IPTV dataset).

3.1.1. Implementation

The authors have a repository containing a library of methods for learning point processes implemented using Matlab. Unfortunately, there are no Matlab experts among our team. We provide a PyTorch implementation of DMMHP in our repository. As reproducing the approach completely is not our main goal our implementation has several specifications. First, we don't reproduce the tuning of basic function parameters as described in the original paper. Second, we encountered a problem when computing negative log-likelihood because as we understand there is no straightforward way to compute it for Hawkes mixture without computing likelihood directly. But when doing it we need to compute the product of many (length of sequence) items and therefore we get an overflow error. Thus in our implementation, we skip all the steps using the computation of negative log-likelihood.

3.2. Deep Learning methods for Unsupervised Learning

Cohortney algorithm allows us to obtain partitions and then find the similarity between different time-series, comparing their partitions. However, this method doesn't take into account that point-processes with different partitions can be from the same cluster and also that there can be some hidden similarities that in general can be described with intensity function.

To partly overcome these issues we propose to use different deep learning techniques on top of Cohortney partitions and assume that Neural Networks can catch these hidden similarities. To do so we consider autoencoders that have been proven to be good in representation learning and also adjust two methods (Guo et al., 2017) and (Asano et al., 2020) that has shown great performance on image clustering.

More about these methods will be said in Algorithms and Models as soon they have been used for learning.

4. Algorithms and Models

4.1. Cohortney

In this section, we have implemented two algorithms presented in Cohortney manuscript. In order to make the text below more clear let's introduce some parameters, we are going to use. The first parameter is an event sequence p (in this work we consider one post to be an event):

$$p = (a_1, a_2, \dots, a_n), \tag{8}$$

where a_i - is a time when the comment under the post appears. We have a bunch of such posts, our source data:

$$P = (p_1, p_2, \dots, p_n). \tag{9}$$

We need to clusterize all these events from P into different clusters. Let there be the parameter t, which we call the present moment. This parameter is fixed, and no comment is considered to be posted after t: $a_n \leq t$. We have an an arbitrary time interval (t_1, t_2) and n events (comments) may happen in this interval. Thus we can introduce the weight function:

$$fws(p, t_1, t_2) = \min([\log_2(n+1)], 9). \tag{10}$$

Now let's define a weight function for the entire time sequence. Let's call the partition ΔT^* on the half interval $(0, t]$ the function with the following form:

$$\Delta T^* = \{t_0, t_1, \dots, t_m\}, 0 = t_0 \leq t_1 \leq \dots \leq t_m = t \tag{11}$$

The distance between the adjacent elements is Δt_n which can be computed as:

$$\Delta t_n = \frac{T^*}{2^n} \tag{12}$$

Hence the weight function for the entire time sequence can be computed the following way:

$$fw(p, t, \Delta T^*) = \{fws(p, t_0, t_1), \dots, fws(p, t_{m-1}, t_m)\} \tag{13}$$

After this brief introduction let's introduce two algorithms: Cohort1, and Cohort2. Depending on time partition ΔT_J and current moment T_j cohort1 computes triplets for each event from P. Then, based on triplets, we put an event to a specific cluster, one event may be in several classes. We arrange this using `dict` structure in Python.

Cohort2 computes a bunch of triplets for a new incoming event, then it compares its triplet to the already obtained triplets in order to find the closest cluster to this new event.

COHORT 1

In this algorithm, we get the set of clusters from the bunch of events.

Algorithm 1 COHORT1

```
Data: $P$
Result: Set of clusters
initialization for $T_j$ in grid do
    Compute the partition $\Delta T_j^n$ for $p_k$ in $P$ do
        Compute the triplet $(T_j, \Delta T_j^n, fw(p_k, \Delta T_j^n)$
         Group the $p_k$ by it's triplet if cluster size ¡ N then
            Dicard this cluster
        end
    end
end
```

COHORT2

Algorithm 2 COHORT2

```
Data: $p_i, t$
Result: Nearest cluster
Find the nearest node $T_t$ to $t$ that does not exceed $t$ for $i$ in $n$ do
    Compute the triplet $(T_j, \Delta T_j^n, fw(p_k, \Delta T_j^n)$
end
for $triplet\ in\ clusters$ do
    n = 0 Find the closest cluster Group the $p_k$ by it's triplet n = n+1
    if no cluster then
        Take n-1 as a closest cluster
    end
end
```

In this algorithm, we find the closest cluster from the already existed clusters for a new event sequence.

As you can see both these algorithms are vague and leave room for speculations, ideas. This Cohort part will be modified in the next iterations of our projects. For the first iteration of our project, we have implemented a verbatim model in order to understand how these algorithms work.

So the first implementation is a cyclic implementation with brute-force. We used $ticks$ library in order to implement a synthetic toy dataset to tune the baseline. From our synthetic dataset, we calculated a triplet for every event. And thus, on every iteration, we get a bunch of n triplets for every event p. In order not to produce any clustering algorithm after the cohort1 we decided to use a dictionary structure in python, which allowed us to use a specific triplet as a key and set of the events with this triplet as a value of this. Thus we automatically receive the clusters. After the Cohort1 finishes, its work we discarded all the clusters, where we have less than N events.

For the Cohort2 we should generate a new event, which we generate with `tick` library. We randomly choose the value of t. Then we compute triplets for each possible partition. Then, by brute-force, we find if we compare our clusters to the clusters we obtained from Cohort1. That's how we find the closest or the coinciding cluster for our new event. This spot is now implemented by word, but I suppose we may increase the speed of this cycle by applying any sorting algorithm for strings. The task of finding the nearest cluster in these terms may be reduced to string sorting. This idea, besides the deep implementation of this part, will be checked in the next iteration of our project.

4.2. Clustering with Convolutional Autoencoders

One of the ideas, how to cluster the sequences using COHORTNEY was deep clustering with convolutional autoencoders. This well-known and widely used method was offered in (Guo et al., 2017). In the original paper (Guo et al., 2017) the clustering process was performed on 3D tensors (pictures) and 2D convolutions were used. A conventional autoencoder is generally composed of two layers, corresponding to encoder $f_w()$ and decoder $g_U()$ respectively. It aims to find a code for each input sample by minimizing the mean squared errors (MSE) between its input and output over all samples, i.e.

$$\min_{W,U} \frac{1}{n} \sum_{1}^{n} \|g_u(f_W(x_i)) - x_i\|_2^2 \tag{14}$$

For fully connected autoencoder we obtain

$$\begin{gathered} f_W(x) = h \\ g_U(h) = \hat{x} \end{gathered} \tag{15}$$

Where h is latent feature representaion, $\hat{x}$ is model prediction. Convolutional Autoencoders (CAE) were trained in end-to-end manner for learning features in unlabeled images.

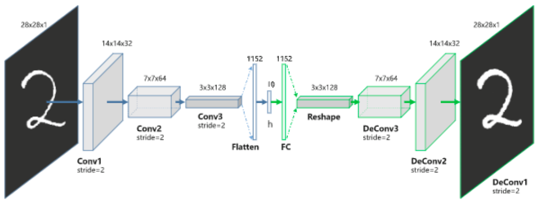

Figure 1. Convolutional AutoEncoders (CAE) for MNIST.

Then, a clustering oriented loss is directly built on embedded features to jointly perform feature refinement and cluster assignment.

In our case, we have 1D tensors of the weight function 13. We decided to use 1D Convolutional autoencoders in order to get latent features and cluster them with the KMeans algorithm. The basic idea was to train a 1D convolutional autoencoder with MSE 14 loss and then to get latent feature representation from the encoder. As we have multiclass sequences in our IPTV and simulated Hawkes datasets, the initial amount of classes was considered as the input features. Then, after getting the latent features from the encoder, we pass the reshaped latent data to the Kmeans algorithm.

4.2.1. Implementation

We have implemented a 1D convolutional autoencoder using `Pytorch Lightning` framework. This framework is a wrapper over the `Pytorch` in order to speed up this library. The structure of the convolutional autoencoder is straightforward: it consists of 5 1D convolutions with Batch Normalization in between. We used the default setting for torch convolutional layer: stride equals to 1 and padding equals 0. Kernel size was set to 3. In order to cluster the encoder output we used standard `sklearn` library with `kmeans` algorithm.

4.3. Deep Clustering

Given partitions obtained from Cohortney for a particular n we may proceed with clustering algorithms suitable for sequences of features of the same length. Therefore we try to combine Cohortney with a couple of deep learning approaches for object clustering. The first one we describe here is a method for image clustering proposed in (**?**) that is called Deep Clustering.

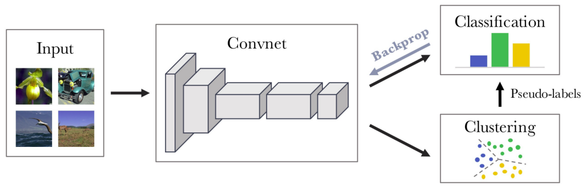

Figure 2. Illustration of the Deep Clustering

The basic idea of the approach is described in the Figure 2. We denote by f_θ the convnet mapping, where θ is the set of corresponding parameters. We refer $f_\theta(x)$ to the vector obtained by applying this mapping to a sequence. Given a training set $X = \{x_1, x_2, \ldots, x_N\}$ of N sequences (in Cohortny partition representation f_w), we want to find a parameter θ^* such that the mapping f_{θ^*} produces good general-purpose features.

These parameters are traditionally learned with supervision, i.e. each sequence x_n is associated with a label y_n in $\{0,1\}^k$. This label represents the image's membership to one of k possible predefined classes. A parametrized classifier g_W predicts the correct labels on top of the features $f_\theta(x_n)$. The parameters W of the classifier and the parameter θ of the mapping are then jointly learned by Deep Clustering for Unsupervised Learning of Visual Features optimizing the following problem:

$$\min_{\theta, W} \frac{1}{N} \sum_{n=1}^{N} \ell\left(g_W\left(f_\theta\left(x_n\right)\right), y_n\right) \tag{16}$$

where ℓ is the multinomial logistic loss, also known as the negative log-softmax function. This cost function is minimized using mini-batch stochastic gradient descent and backpropagation to compute the gradient.

More specifically the training consists of the following steps. On each epoch of the training phase, deep features of sequences are computed with convnet. These features are used to cluster sequences using the KMeans algorithm. The top fully-connected classifier is then initialized and put onto the top of the model. The obtained pseudolabels are then used for supervised-manner training of the model.

4.3.1. Implementation

We adopt the original implementation of the Deep Clustering. We use `sklearn` implementation of KMeans algorithm with `kmeans++` initialization. We standardize features with sklearn realization of `StandardScaler` before clustering. One of the drawbacks of this approach we observed is that there is no guarantee of convergence of assigned clusters and in fact, the quality of clustering may change dramatically during training. As a consequence of this feature is that the variation of the clustering quality may be huge. Besides we observe that the algorithm is highly sensitive to change of hyperparameters.

4.4. Optimal Transport

In this work, we adopted a state-of-the-art approach for image unsupervised classification proposed in (Asano et al., 2020). The idea of the approach is to do representation learning and clustering at the same time. It has shown to be effective for images and in this work we are using the same technique for event sequences clustering.

So let's consider the problem of unsupervised classification for some instances I. Let's consider the NN combing the representation NN $x = \Phi(I)$ and a classification head h. So x will be a feature vector. Each data point $I_1, \ldots, I_N$ has corresponding label $y \in \{1, \ldots, K\}$. The model should map from the data points space to the scores of each class, which can be considered as probability after applying a softmax operator.

$$p(y = \cdot | \mathbf{x}_i) = \text{softmax}(h \circ \Phi(\mathbf{x}_i)) \tag{17}$$

In this case the cross-entropy loss will be:

$$E(p|y_1,\ldots,y_n) = -\frac{1}{N}\sum_{i=1}^{N}\log p(y_i|\mathbf{x}_i) \tag{18}$$

For semi-supervised classification one can consider optimizing loss both with respect to the model and to the labels. However, in the case of unsupervised learning, this will lead to a degenerate solution. So it was proposed to introduce the posterior distribution $q(y|\mathbf{x}_i)$ and to additionally constraint the clustering with equipartition. Then the problem can be written as follows:

$$\begin{gathered} E(p,q) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{y=1}^{K} q(y|\mathbf{x}_i)\log p(y|\mathbf{x}_i) \\ \min_{p,q} E(p,q) \text{ subject to} \\ \forall y: q(y|\mathbf{x}_i) \in \{0,1\} \text{ and } \sum_{i=1}^{N} q(y|\mathbf{x}_i) = \frac{N}{K} \end{gathered} \tag{19}$$

Let's denote $P_{yi} = p(y|\mathbf{x}_i)\frac{1}{N}$ and $Q_{yi} = q(y|\mathbf{x}_i)\frac{1}{N}$. Let's relax matrix Q to be an element of the transportation polytope:

$$U(r,c) = \{Q \in \mathbb{R}_{+}^{K\times N} | Q\mathbf{1} = r, Q^\top \mathbf{1} = c\} \tag{20}$$

Vector $\mathbf{1}$ denotes all ones of appropriate dimensions.

$$r = \frac{1}{K}\mathbf{1},\ c = \frac{1}{N}\mathbf{1} \tag{21}$$

In this case the objective function can be rewritten as:

$$E(q,p) + \log N = \langle Q, -\log P\rangle \tag{22}$$

So the optimization is equivalent to solving the minimization problem:

$$\min_{Q\in U(r,c)} \langle Q, -\log P\rangle \tag{23}$$

This is a transport problem and this problem can be solved in polynomial time, however for big datasets, it can be inefficient, that's why it was proposed to use a fast Sinkhnorn-Knopp algorithm. In this case, the objective function also has a regularization term:

$$\min_{Q\in U(r,c)} \langle Q, -\log P\rangle + \frac{1}{\lambda}\mathrm{KL}(Q||rc^\top) \tag{24}$$

Minimizer of the equation is $Q = \mathrm{diag}(\alpha)P^\lambda\mathrm{diag}(\beta)$. Here the exponentiation is elementwise and vectors α and β a chosen so that the resulting matrix was a probability matrix. Choosing λ is a tradeoff between the convergence speed and the closeness to the original problem.

So the optimization consists of two steps. For a given label assignment optimizing the model with respect to the parameters. And then given the probabilities P update iteratively α and β with the following formulas:

$$\forall y: \alpha_y \leftarrow [P^\lambda \beta]_y^{-1}, \forall i: \beta_i \leftarrow [\alpha^\top P^\lambda]_i^{-1} \tag{25}$$

4.4.1. Implementation

The code from (Asano et al., 2020) has been used as a reference. To train the model the following approach has been used. Given the maximal time T_h the following partitions has been used:

$$(T_j, T_j(T_j/2^k), fws),\ T_j = T_h/2^n \tag{26}$$

Here $k,\ n$ take values from 0 to some hyper-parameters $K,\ N$. Then we train convolutional filters followed by linear layer and predicting linear layer on top of these partitions.

5. Experiments and Results

5.1. Data

For our experiments, we used the real-world Internet Protocol Television (IPTV) dataset of 300 event sequences - one per user - and a range of synthetic datasets collected using the library for simulating Hawkes processes `tick`. You may download the IPTV dataset via the following link: IPTV_Data.zip

Following (Xu & Zha, 2017) we generated 5-dimensional realizations of Hawkes processes for 3, 4, 5 clusters (3 datasets). We generated 400 realizations per each cluster. You may download all synthetic datasets via the following link: simulated_Hawkes.zip

Here we place the snippet of `Python` code that we use for data generation:

```
from tick.hawkes import
    SimuHawkesExpKernels

def simulate_hawkes(n_nodes, n_realiz,
    end_time, dt, max_jumps=1000, seed=None
    , adj_density=0.25):
    """
    Simulate Hawkes process realizations

    Args:
        n_nodes (int)        - dimension of
    Hawkes process
        n_realiz (int)       - number of
    realizations
```

```
        end_time (float)     - right edge of
     interval where timestamps are
    distributed
        dt (float)           - delta to
    track intensity
        max_jumps (int)      - maximum
    number of jumps in Hawkes process
        seed (int)           - random seed
        adj_density (float) - density of
    the adjacency matrix of Hawkes process
    intensity

    Return:
        tss (List)           - list of
    realizations
    """

    tss = []

    baselines = np.random.rand(n_nodes) /
    n_nodes
    decays = 5 * np.random.rand(n_nodes,
    n_nodes)
    adjacency = spsp.rand(n_nodes, n_nodes,
     density=adj_density).toarray()

    for i in range(n_realiz):
        seed_ = seed + i if seed is not
    None else None
        hawkes = SimuHawkesExpKernels(
            baseline=baselines, decays=
    decays, adjacency=adjacency, seed=seed_
    )
        hawkes.adjust_spectral_radius(0.8)
        hawkes.max_jumps = max_jumps

        hawkes.end_time = end_time
        hawkes.verbose = False

        hawkes.track_intensity(dt)
        hawkes.simulate()
        tss.append(hawkes.timestamps)

    return tss
```

Listing 1. Snippet of synthetic data generation

5.2. Evaluation

For our experiments, we adopt the evaluation proposed in (Xu & Zha, 2017). As for the IPTV dataset, there are no real labels it is not possible to measure the correctness of clustering. Therefore authors propose to measure the clustering consistency via a cross-validation method. The principle is simple: because random sampling does not change the clustering structure of data, a clustering method with high consistency should preserve the pairwise relationships of samples in different trials. We test each clustering method with $J(=10)$ trials. In the j-th trial, data is randomly divided into two folds. After learning the corresponding model from the training fold, we apply the method to the testing fold. We enumerate all pairs of sequences within the same cluster in the j -th trial and count the pairs preserved in all other trials. The clustering consistency is the minimum proportion of preserved pairs over all trials:

$$\text{Consistency} = \min_{j\in\{1,\dots,J\}} \sum_{j'\neq j} \sum_{(n,n')\in\mathcal{M},} \frac{1\left\{k_n^{j'} = k_{n'}^{j'}\right\}}{(J-1)\left|\mathcal{M}_j\right|},$$

where $\mathcal{M}_j = \left\{(n,n') \mid k_n^j = k_{n'}^j\right\}$ is the set of sequence pairs within same cluster in the j -th trial, and k_n^j is the index of cluster of the n -th sequence in the j -th trial.

For the synthetic data with clustering labels, we use clustering purity to evaluate various methods:

$$\text{Purity} = \frac{1}{N}\sum_{k=1}^{K} \max_{j\in\{1,\dots,K'\}} \left|\mathcal{W}_k \cap \mathcal{C}_j\right|$$

where $\mathcal{W}_k$ is the learned index set of sequences belonging to the k -th cluster, $\mathcal{C}_j$ is the real index set of sequence belonging to the j -th class, and N is the total number of sequences.

5.3. Implementation details

In all our experiments with DMHP we set the number of outer iterations equal to 10. The number of inner iterations grows from 2 to 8 for the first 7 iterations and stays 8 for the last 3 iterations. We took 5 Gaussian basis functions with different scales. The replication of experiments with DMHP can be found in our github.

For Deep Clustering we use a simple convolutional neural network with batch normalizations. For the main part of the model we use Adam optimizer and for the top layer, we use SGD optimizer with a momentum equal 0.9. We train the model for 30 epochs with learning rate=1e-4, weight decay=1e-4. We apply Deep Clustering over features obtained from Cohortney partition with $n = 8$. All our experiments were run on the CPU.

For CAE Clustering we use a straightforward convolutional autoencoder with 5 convolutions and batch normalizations between the convolutions. The kernel size was set to 3, stride and padding were default. As a training criterion Mean Squared Error (14) was chosen. We took Adam optimizer with standard parameters. We trained Convolutional Autoencoder with Pytorch Lightning framework on 50 epochs. Batch size equals to 100. We applied CAE Clustering over features obtained from Cohortney partition with $n = 8$. The number of out channels from the encoder (the size of latent features space) is 12 for all the datasets. CAE was trained on GPU.

5.4. Results

To evaluate methods on the IPTV dataset we ran them for 10 trials to collect assigned labels and compute Consistency. The result is presented in Table 1.

Table 1. Clustering Consistency on Real-world Data

method	**DMHP**	**Deep Clustering**	**CAE Clustering**
IPTV data	0.3660	0.4234	**0.4241**

We computed the mean and the standard deviation of Purity over 5 runs. The result is presented in Table 2. We denote K as a number of clusters and C as a number of classes.

We mark results bold when they are *better* than ones of alternative methods.

For CAE clustering we have a parameter we can control - the number of encoder out channels or the size of latent feature space. In order to investigate if the size of latent features space impacts clustering quality, we hold several experiments and average the results. All the experiments were held on the dataset which showed the worst performance on CAE clustering - Synthetic dataset $K = 5$, $N = 5$.

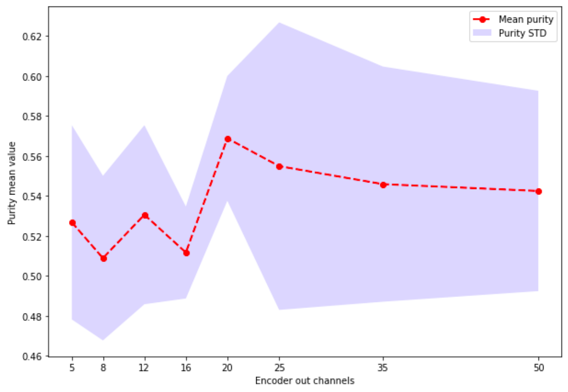

Figure 3. The dependency of purity from the size of latent feature space

We held 5 launches for different sizes of latent feature spaces. For multiclass simulated Hawkes dataset $K = 5$, $C = 5$ with five clusters the number of encoder out channels were $[5, 8, 12, 16, 20, 25, 35, 50]$. As you can on the 3 the average purity is set about 0.536 and does not chanfe significantly. Thus we can conclude that the size of latent features space impacts lightly on the clustering quality.

6. Conclusion

We could not beat a baseline model for synthetic Data, but we could come up with better results on real world IPTV dataset. All the proposed models came up with adequate results, and these models are non-parametrical, what is advantage. We don't need process to be Hawkes process type for models to show the good performance. In further research the model can be improved if one takes the nature of the process into account and if one computes intensity directly.

References

Asano, Y. M., Rupprecht, C., and Vedaldi, A. Self-labeling via simultaneous clustering and representation learning. *Conference paper at ICLR 2020*, 2020.

Guo, X., Liu, X., Zhu, E., and Yin, J. Deep clustering with convolutional autoencoders. In *International conference on neural information processing*, pp. 373–382. Springer, 2017.

Krueger, R., Vij, A., and Rashidi, T. H. A dirichlet process mixture model of discrete choice, 2018.

Liao, T. Clustering time series data — a survey. *Pattern Recognition*, 38:1857–1874, 11 2005. doi: 10.1016/j.patcog.2005.01.025.

Xu, H. and Zha, H. A dirichlet mixture model of hawkes processes for event sequence clustering. In *Advances in Neural Information Processing Systems*, pp. 1354–1363, 2017.

Table 2. Clustering Purity on Synthetic Data

method	**DMHP**	**Deep Clustering**	**CAE Clustering**	**Optimal Transport Clustering**
K=5, C=5	**.6108**$\pm$.0495	.4384$\pm$.0985	.5306$\pm$.0447	.4537$\pm$.0513
K=4, C=5	**.7505**$\pm$.0212	.5613$\pm$.0158	.7113$\pm$.0474	.5372$\pm$0.0295
K=3, C=5	**.8723**$\pm$.0298	.6945$\pm$.1023	.797$\pm$.0581	.6731$\pm$0.8203

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Vladislav Zhuzhel (25% of work)

- Implementation of Neural Hawkes model (was not needed in final version)
- Implementation of Sinkhorn-Knopp Algorithm for Optimal Transport Problem
- Implementation of multiclass for Optimal Transport Problem + experiments
- Preparing Section 1, 4.4, 3 of this report

Evgeny Lagutin (25% of work)

- Implementation of Dirichlet Mixture Model and metrics from (Xu & Zha, 2017) + experiments
- Implementation of Deep Clustering + experiments
- Implementation of synthetic data generation
- Preparing Sections 2, 4.3, 3.1, 5 of this report
- Preparing final presentation

Anna Rudenko (25% of work)

- Implementation of COHORTNEY + experiments
- Implementation of CAE + experiments
- Preparing Sections 4.2, 4.1, 5, 6 of this report

Liliya Mironova (25% of work)

- Implementation of Soft-DTW
- Implementation of Optimal Transport (adaptation of Asano self-labeling code)
- Preparing Sections 2, 4.4, 1 and the general structure of this report
- Preparing final presentation
- Recording a presentation video

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: 90% of our code was written in our own, we adopt the code of Deep Clustering from the original implementation.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: No data were excluded, no standard or specific pre-processing was done

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☐ Yes.
   ☐ No.
   ☑ Not applicable.

   Students' comment: Neither validation nor testing are suitable for this work

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    ☑ Yes.

- ☐ No.
- ☐ Not applicable.

Students' comment: None

13. A description of the computing infrastructure used is included in the report.

    - [✓ Yes.]
    - ☐ No.
    - ☐ Not applicable.

    Students' comment: None