Balanced Jump Model for Time Series Data

Yuqi Nie
Department of Electrical and Computer Engineering
Princeton University
Princeton, NJ 08544
ynie@princeton.edu

Abstract

Fitting hidden states in time series could be challenging without prior knowledge about data. For instance, hidden Markov model (HMM) is not adequate for fitting states when self-transition probability is close to 1, while standard jump model is less effective when self-transition probability is low. In this paper, I propose a balanced jump model fitting algorithm combining the properties of both HMM and standard jump model under the framework of general jump model. It not only outperforms these original models, but also shows great capability of generalization on different self-transition probabilities. This method could be very useful in many practical time series applications, and my code is available on GitHub [1].

1 Introduction

Data clustering is essential in time series analysis, especially when data is considered to have properties based on latent states, which is common in many fields such as regime-switching model in stock market [2]. This can be understood as labels/modes in classification tasks or hidden states in hidden Markov model.

The main goal of this project is to develop a better algorithm that combining the advantages of current methods to find the hidden states in time series data. Especially, I would like to focus on modifying the standard jump model penalty based on hidden Markov model under the framework of general jump model.

Assume we observe a set of data $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ..., \mathbf{x}_T)$ with the corresponding hidden states $\mathbf{S} = (s_1, s_2, ..., s_t, ..., s_T)$, where the states $s \in \{1, ..., k\}$ are parameterized by their centroid $\{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k\}$. I will introduce some popular algorithms to identify the states in time series.

1.1 K-means Clustering

K-means clustering is a widely used algorithm which clusters the observed vectors effectively with expectation–maximization (EM) algorithm. The objective is:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_t \in A_i} ||\mathbf{x}_t - \boldsymbol{\mu}_i||^2 \tag{1}$$

Here $A_i = \{\mathbf{x}_t : s_t = i\}$ is the data set that assigned to the state label i with centroid $\boldsymbol{\mu}_i$. The EM algorithm first fix the state label and tune $\boldsymbol{\mu}_i$ to minimize the objective function, then fix these centroids and re-assign the states.

[1] https://github.com/machinelearninggo/COS-513-Project. Some jump model codes credit to [1].

Preprint. Under review.

1.2 Hidden Markov Model (HMM)

In a HMM, the hidden states can take can take discrete values, and the sequence $\mathbf{S}$ satisfies the Markov property: $p(s_t|s_{t-1}, ...s_1) = p(s_t|s_{t-1})$. Then $\mathbf{x}_t$ is drawn from a distribution parameterized by hidden states. Suppose that HMM is parameterized by Θ, the maximum a posteriori (MAP) is:

$$\hat{\Theta} = \arg\max_{\Theta}[p(\Theta|\mathbf{X})] = \arg\max_{\Theta}[p(\mathbf{X}|\Theta)p(\Theta)] \tag{2}$$

If we don't have prior info about Θ, then $p(\Theta)$ is uniform distribution, and the MAP becomes the same as usual machine learning estimation, which can be done using EM algorithms [3]. In general, HMM are usually trained using Baum-Welch algorithm, which is a special case of the EM algorithm making use of the forward-backward algorithm [4].

1.3 Jump Model

Jump model is a simple but effective idea of learning latent variables and clustering time series data. Surprisingly, it was first proposed in 2018 to the best of my knowledge [4], which is fairly new. The key idea is to penalize the objective function when there is a change in hidden states [5]:

$$J'(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S}) = \{l(\mathbf{x}_1, \boldsymbol{\mu}_{s_1}) + \sum_{t=2}^{T}[l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t}) + \lambda \mathbb{I}_{s_t \neq s_{t-1}}]\} \tag{3}$$

Intuitively, minimizing $J'(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S})$ is equivalent to trying to avoid state changes. Here the loss function is usually picked as $l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t}) = ||\mathbf{x}_t - \boldsymbol{\mu}_{s_t}||^2$, but can also be designed according to the tasks.

Jump model has some variants, for example the sparse jump model which is inspired from sparse k-means clustering and combines the purpose of selecting features as well as estimating states/parameters [1, 6]. There is also a more expressive and generalized framework of jump model which we will see in the next section.

2 HMM as General Jump Model

As an extension of Eq. (3), the objective of general jump model which we want to minimize is [4]:

$$J(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S}) = \sum_{t=1}^{T} l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t}) + \sum_{k=1}^{K} r(\boldsymbol{\mu}_k) + \mathcal{L}(\mathbf{S}) \tag{4}$$

Here $\mathbf{S} = (s_1, s_2, ..., s_t, ..., s_T)$ is the state sequence, $r(\boldsymbol{\mu}_k)$ is the loss only depends on model parameters, and the last term defines the loss penalty related to the states and transitions:

$$\mathcal{L}(\mathbf{S}) = \mathcal{L}^{\text{init}}(s_1) + \sum_{i=2}^{T} \mathcal{L}^{\text{state}}(s_t) + \sum_{i=2}^{T} \mathcal{L}^{\text{trans}}(s_t, s_{t-1}) \tag{5}$$

The first term $\mathcal{L}^{\text{init}}(s_1)$ is the initial state cost, the second term $\mathcal{L}^{\text{state}}(s_t)$ is the cost in the later states, and the last term $\mathcal{L}^{\text{trans}}(s_t, s_{t-1})$ is then state transition cost. So easily we can see from Eq. (3) that in a standard jump model has $r(\boldsymbol{\mu}_k) = \mathcal{L}^{\text{init}}(s_1) = \mathcal{L}^{\text{state}}(s_t) = 0$ and $\mathcal{L}^{\text{trans}}(s_t, s_{t-1}) = \lambda \mathbb{I}_{s_t \neq s_{t-1}}$.

It is proven in [4] that HMM can be unified under the framework of general jump model. In HMM, suppose the transition probabilities and the initial state distribution are:

$$p(s_{t+1} = i|s_t = j) = \pi_{i,j}, \quad i, j \in (1, ..., k) \tag{6a}$$

$$p(s_1) = \pi_{s_1} \tag{6b}$$

If we set:

$$l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t}) = -\log p(\mathbf{x}_t|\boldsymbol{\mu}_{s_t}) \tag{7a}$$
$$r(\boldsymbol{\mu}_k) = 0 \tag{7b}$$
$$\mathcal{L}^{\text{init}}(s_1) = -\log \pi_{s_1} \tag{7c}$$
$$\mathcal{L}^{\text{state}}(s_t) = 0 \tag{7d}$$
$$\mathcal{L}^{\text{trans}}(s_t, s_{t-1}) = -\log \pi_{s_t, s_{t-1}} \tag{7e}$$

then minimizing this ojective function is equivalent to solving a HMM model. A special case is Gaussian HMM, where $p(\mathbf{x}_t|\boldsymbol{\mu}_{s_t})$ is a normal distribution function, thus the loss is $l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t}) = c + \frac{1}{2\sigma^2}||\mathbf{x}_t - \boldsymbol{\mu}_{s_t}||^2$, which has the same form as k-means clustering or standard jump model. But the penalty term $-\log \pi_{s_t, s_{t-1}}$ is not a constant which is different with the standard jump model.

We conclude that general jump model can express HMM, and its jump penalty is $-\log \pi_{s_t, s_{t-1}}$, which indicates a lower penalty when the inter-state transition probability is high, i.e. a low self-transition probability. This phenomenon inspires our discussion in the next section.

3 Balanced Penalty in Jump Model

We have seen in the previous sections that the transition penalty of standard jump model has constant coefficient and just depends on whether the state changes. This would work well when self-transition probability is high, but may cause trouble in the opposite case. On the other hand, HMM provides a form of penalty which is proportional to the log of inter-state transition probability. It performs well in most of the cases, but when self-transition probability is high, the fitting accuracy will drop, which is shown in table 1 and section 5.

Here is a possible reason. Since HMM can be expressed within the framework of general jump model, it is easy to see that the goal of minimizing the objective function in Eq. (4) will also minimize transition penalty $\mathcal{L}^{\text{trans}}(s_t, s_{t-1}) = \log \frac{\pi_{s_{t-1}, s_{t-1}}}{\pi_{s_t, s_{t-1}}}$ for state changes. When self-transition probability $\pi_{s_{t-1}, s_{t-1}}$ is very close to 1, $\pi_{s_t, s_{t-1}}$ will be small, which leads to a large penalty. Thus HMM will avoid to fit it in such a way. But in reality, sometimes we do have a very high self-transition probability, then HMM will gain less accuracy than standard jump model.

We can't always be guaranteed to have prior knowledge about the scale of self-transition probability. It is natural to think about: can we combine those two models under the general jump model framework to have a more subtle penalty term which could deal with a general time series case?

The answer is yes, and we use an auto-control parameter θ_t to achieve it. We call it a balanced jump model, and its objective function can be expressed as:

$$J_B(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S}) = \left\{ C \sum_{t=1}^{T} l(\mathbf{x_t}, \boldsymbol{\mu}_{\boldsymbol{s_t}}) - \sum_{t=2}^{T} \left[(1-\theta_t) * \mathcal{L}_{HMM}(s_t) + \theta_t * \mathcal{L}_{Jump}(s_t) \right] \right\} \tag{8}$$

Where $\mathcal{L}_{HMM}(s_t)$ is the penalty inspired from HMM, and $\mathcal{L}_{Jump}(s_t)$ is the standard jump penalty.

$$\theta_t = \tanh\left(\frac{\pi_{s_{t-1}, s_{t-1}}}{\pi_{s_t, s_{t-1}}}\right) \tag{9a}$$
$$\mathcal{L}_{HMM}(s_t) = -\log \pi_{s_t, s_{t-1}} \tag{9b}$$
$$\mathcal{L}_{Jump}(s_t) = \lambda \mathbb{I}_{s_t \neq s_{t-1}} \tag{9c}$$

We ignore the initial state penalty here since it is negligible when sequence is long. So if self-transition probability $\pi_{s_{t-1}, s_{t-1}}$ is very close to 1, θ_t will also be close to 1, which indicates an emphasizing on standard jump penalty and ignoring the HMM penalty. But when $\pi_{s_{t-1}, s_{t-1}}$ is lower, more HMM penalty will be added to the objective function. In this way we gain a balance between the standard jump model and HMM.

4 Algorithms

There are 2 hyper-parameters C, λ that need to be determined when fitting models. Besides doing direct grid search, we can first determine the jump penalty parameter λ by searching on standard jump model, then search the hyper-parameters C on the balanced jump model. Thus we can write out the meta fitting algorithm, standard jump model fitting algorithm and balanced jump model fitting algorithm respectively:

Algorithm 1 Meta Fitting Algorithm

Input: sequences of train data $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ..., \mathbf{x}_T)$, hyper-parameter set Φ and objective function $J(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S}, \Phi)$
Output: model parameters $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_k\}$ and state sequence $\mathbf{S} = (s_1, s_2, ..., s_t, ..., s_T)$
initialize $\mathbf{S}^0$;
while $\mathbf{S}^j \neq \mathbf{S}^{j-1}$ **do**
  $\boldsymbol{\mu}^j = \arg\min \sum_{t=1}^{T} [l(\mathbf{x}_t, \boldsymbol{\mu}_{s_t^{j-1}})] = \arg\min \sum_{t=1}^{T} ||\mathbf{x}_t - \boldsymbol{\mu}_{s_t^{j-1}}||^2$
  $\mathbf{S}^{j-1} = \arg\min J(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S}, \Phi)$
end

Algorithm 2 Fitting Standard Jump Model

Input: Set of train data $\{\mathbf{X}\}$, set of test data $\{\mathbf{X}'\}$ and objective function $J'(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S})$
Output: model parameters $\boldsymbol{\mu}$, hyper-parameters $\lambda^\star$ and testing accuracy p'
for λ *in range(*λ_{min}*,* λ_{max}*, step*$_\lambda$*)* **do**
  for $\mathbf{X}_i \in \{\mathbf{X}\}$ **do**
    do algorithm 1 with $\mathbf{X}_i$, λ and $J'(\mathbf{X}_i, \boldsymbol{\mu}, \mathbf{S})$;
  end
  get average fitting accuracy on $\{\mathbf{X}\}$;
end
select the best $\lambda^\star$;
for $\mathbf{X}'_i \in \{\mathbf{X}'\}$ **do**
  do algorithm 1 with $\mathbf{X}'_i$, $\lambda^\star$ and $J'(\mathbf{X}', \boldsymbol{\mu}, \mathbf{S})$;
end
get average testing accuracy p' on $\{\mathbf{X}'\}$;

Algorithm 3 Fitting Balanced Jump Model

Input: Set of train data $\{\mathbf{X}\}$, set of test data $\{\mathbf{X}'\}$ and objective function $J'(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S})$, $J_B(\mathbf{X}, \boldsymbol{\mu}, \mathbf{S})$
Output: model parameters $\boldsymbol{\mu}$, hyper-parameters $C^\star, \lambda^\star$ and testing accuracy p_B
for λ *in range(*λ_{min}*,* λ_{max}*, step*$_\lambda$*)* **do**
  for $\mathbf{X}_i \in \{\mathbf{X}\}$ **do**
    do algorithm 1 with $\mathbf{X}_i$, λ and $J'(\mathbf{X}_i, \boldsymbol{\mu}, \mathbf{S})$;
  end
  get average fitting accuracy on $\{\mathbf{X}\}$;
end
select the best $\lambda^\star$;
for C *in range(*C_{min}*,* C_{max}*, step*$_C$*)* **do**
  for $\mathbf{X}_i \in \{\mathbf{X}\}$ **do**
    do algorithm 1 with $\mathbf{X}_i$, $\Phi = (C, \lambda^\star)$ and $J_B(\mathbf{X}_i, \boldsymbol{\mu}, \mathbf{S})$;
  end
  get average fitting accuracy on $\{\mathbf{X}\}$;
end
select the best $C^\star$;
for $\mathbf{X}'_i \in \{\mathbf{X}'\}$ **do**
  do algorithm 1 with $\mathbf{X}'_i$, $\Phi = (C^\star, \lambda^\star)$ and $J_B(\mathbf{X}'_i, \boldsymbol{\mu}, \mathbf{S})$;
end
get average testing accuracy p_B on $\{\mathbf{X}'\}$;

5 Experiment Results

There are four models tested in our experiments: Gaussian HMM, k-means clustering, standard jump model and balanced jump model. Our train and test data are generated from a two-state multivariate Gaussian HMM:

$$\mathbf{x_t}|s_t \sim \mathcal{N}(\boldsymbol{\mu}_{s_t}, \boldsymbol{\Sigma}_D) \tag{10}$$

where D is the dimension of feature. We set two cases with different transition probabilities of hidden states: case 1 has high self-transition probability, and case 2 has lower self-transition probability. The details of experiments and model parameters can be found in Appendix A, and hyper-parameter tuning can be found in Appendix B.

We here use balanced accuracy (BAC) as a metric following the paper [7]. It is defined as:

$$\text{BAC} = \frac{1}{k}\sum_{i=1}^{k}\frac{tp_i}{tp_i + fn_i} \tag{11}$$

Here tp_i, fn_i are the numbers of true positives and false negatives in state i respectively.

Models	Gaussian HMM	K-means Clustering	Standard Jump Model	Balanced Jump Model
Case 1	0.6677	0.5635	0.9154	**0.9174**
Case 2	0.7605	0.6311	0.7593	**0.7672**

Table 1: BAC in case 1: high self-transition probability; case 2: low self-transition probability.

The results are shown in the table 1. First, it proves our intuition in section 3. When the self-transition probability is high, the standard jump model performs well because it set a high threshold for state transition, which is usually the case in stock market [1]. However when the self-transition probability is low, HMM could be more effective, which is the case in some biological models [8].

Beside, in both high and low self-transition probability cases, we see that the balanced jump model achieves the best BAC, which indicates that it not only benefits from combining penalties from both of the models, but also owns great capability of generalization. This relieves us from making prior assumptions about the self-transition probability of true data, thus could be very useful in many practical time series applications.

6 Conclusion

In this paper, I have proposed a balanced jump model that has great generalization on time series data with high and low self-transition probabilities. This model is inspired from general jump model framework, and by taking a balance between standard jump model and HMM, it not only outperforms them by benefiting from this combination, but also relieves us from obtaining prior knowledge about the self-transition probability of data, which shows the potential to be useful in many practical time series applications.

There are also some further work that remains to be done in future. First, I have only done analysis on synthetic data in this paper. We could use stock data [1], biological data [8] and other real-world data to test the practical performance of our model. Second, we can extend it to a multi-state HMM. Lastly, there are some other HMM variants that allow us to incorporate a preference for longer-lasting states within a generative framework, such as hidden semi-markov model (HSMM) [9]. We can make a comparison or think about how can we combine those models with ours.

Acknowledgement

I appreciate the helpful discussion with Onat Aydinhan, a PhD candidate in Princeton ORFE department. Also I would like to thank Prof. Adji Bousso Dieng's instruction and Assistant Instructors' help in this course.

References

[1] Petter N. Kolm Peter Nystrup and Erik Lindström. Feature selection in jump models. *Expert Systems with Applications*, 184:115558, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.115558. URL https://www.sciencedirect.com/science/article/pii/S0957417421009647.

[2] F.I.A. Mary R. Hardy A.S.A. A regime-switching model of long-term stock returns. *North American Actuarial Journal*, 5(2):41–53, 2001. doi: 10.1080/10920277.2001.10595984. URL https://doi.org/10.1080/10920277.2001.10595984.

[3] Hsiao-Wuen Hon Xuedong Huang, Alex Acero and Raj Reddy. *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR, 2001. URL https://dl.acm.org/doi/abs/10.5555/560905.

[4] Alberto Bemporad, Valentina Breschi, Dario Piga, and Stephen P. Boyd. Fitting jump models. *Automatica*, 96:11–21, 2018. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2018.06.022. URL https://www.sciencedirect.com/science/article/pii/S0005109818303145.

[5] Peter Nystrup, Petter N. Kolm, and Erik Lindström. Greedy online classification of persistent market states using realized intraday volatility features. *The Journal of Financial Data Science*, 2(3):25–39, 2020. ISSN 2640-3943. doi: 10.3905/jfds.2020.2.3.025. URL https://jfds.pm-research.com/content/2/3/25.

[6] Daniela M. Witten and Robert Tibshirani. A framework for feature selection in clustering. *Journal of the American Statistical Association*, 105(490):713–726, 2010. doi: 10.1198/jasa.2010.tm09415. URL https://doi.org/10.1198/jasa.2010.tm09415. PMID: 20811510.

[7] Kay Henning Brodersen, Cheng Soon Ong, Klaas Enno Stephan, and Joachim M. Buhmann. The balanced accuracy and its posterior distribution. In *2010 20th International Conference on Pattern Recognition*, pages 3121–3124, 2010. doi: 10.1109/ICPR.2010.764.

[8] Byung-Jun Yoon. Hidden markov models and their applications in biological sequence analysis. *Current Genomics*, 10(6):402–415, 2009. URL https://dx.doi.org/10.2174/138920209789177575.

[9] Shun-Zheng Yu. Hidden semi-markov models. *Artificial Intelligence*, 174(2):215–243, 2010. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2009.11.011. URL https://www.sciencedirect.com/science/article/pii/S0004370209001416.

A Experiment Settings

Model Parameter and Data Generation. The train and test data sets are generated with a Gaussian HMM described in Eq. (10) using the $hmmlearn$ python package. We generate 100 train data and 100 test data. The sequence length is $T = 1000$, and the dimension of features is $D = 2$. The parameters of case 1 and 2 are:

$$P^1 = \begin{pmatrix} 0.995 & 0.005 \\ 0.002 & 0.998 \end{pmatrix}, \quad P^2 = \begin{pmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{pmatrix} \tag{12a}$$

$$\boldsymbol{\mu}^1 = [[0, 0], [0.4, -0.4]], \quad \boldsymbol{\mu}^2 = [[0, 0], [1, -1]] \tag{12b}$$

$$\Sigma^1 = [\mathbf{I}_2, \mathbf{I}_2], \quad \Sigma^2 = [\mathbf{I}_2, [[2, 0], [0, 1]] \tag{12c}$$

$$\pi_0^1 = \pi_0^2 = (0.6, 0.4) \tag{12d}$$

The $P, \boldsymbol{\mu}, \Sigma, \pi_0$ are transition probability, mean vector, covariance matrix and initial distribution respectively. $\mathbf{I}$ is an identity matrix. More numerical examples besides case 1 and 2 are tested an presented in my GitHub repository.

Gaussian HMM Fitting. The Gaussian HMM is implemented by using the $hmmlearn$ package. The transition matrix prior is set as 1, and the maximum iteration is 100.

Standard Jump Model Fitting Balanced Jump Model Fitting. The original code of jump model credits to [1]. I modify it mainly in the penalty part, and do the hyper-parameter search / testing using algorithm 2 and 3.

K-means Clustering Fitting. K-meaning clustering is a classical algorithm, and we use the $cluster.KMeans$ from $sklearn$ python package. The random state is set as 0.

Evaluation. The evaluation metric is BAC in (11). We use an average BAC over the test data sets with 100 data, as described in algorithm 2 and 3.

B Hyper-parameter Tuning

In this appendix section, I show the visualization of hyper-parameter tuning in both case 1 and 2. The $\lambda^\star$ is used for both algorithm 2 and 3, and $C^\star$ is used for algorithm 3. More numerical examples besides case 1 and 2 are presented in the GitHub repository.

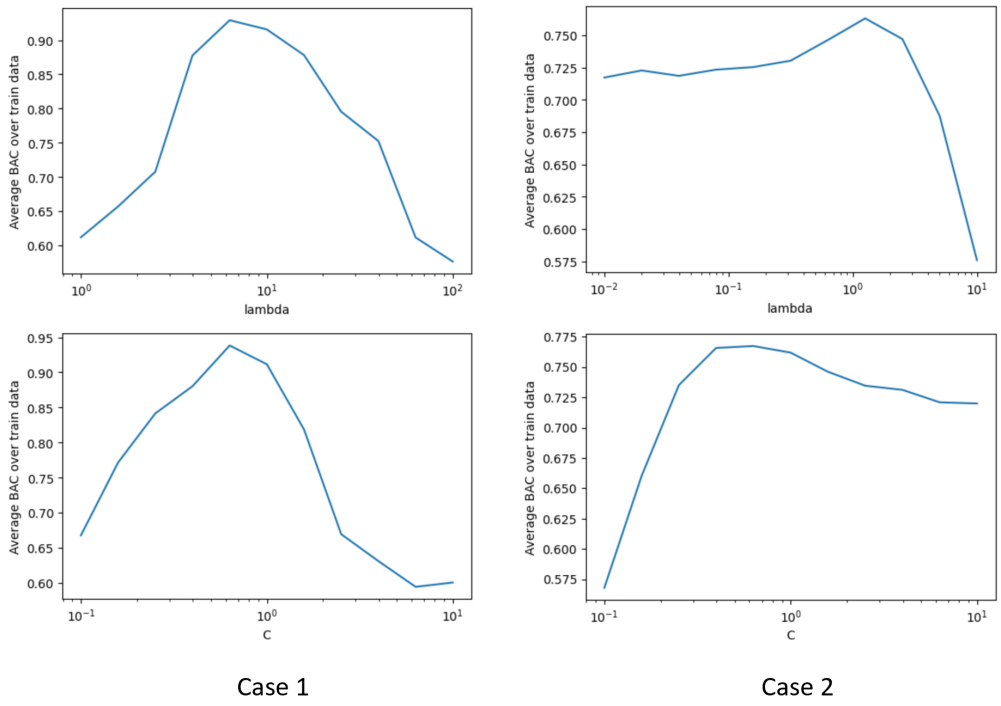

Figure 1: Hyper-parameter search for λ, C. The left column corresponds to case 1, and the right column is case 2.