1. Introduction

In many applications, unlike traditional machine learning where an example is represented by a vector of values, it is more natural to represent the relationship between two examples by a similarity score. In these cases, the data is a matrix. The matrix may have some missing values that indicate that we don’t know the relationship between the two examples or they cannot be directly compared. The matrix may be asymmetric. The relationship between examples may be an ordered relationship. The matrix is also not necessarily square. A common case is: only a certain number of (positives) examples are of interests, the similarity scores between a large number of examples and these examples are computed. Nearest Neighbor method is the most natural and widely used method for these kinds of data.

In this project, we try to apply kernel-learning methods to similarity matrix data. For the sake of simplicity, we assume that the similarity matrix be square. The obstacle we meet is that the similarity matrix is not generally a kernel matrix. As we know, a kernel matrix need to be square, symmetric, semi positive definite and should not contain missing values. A simple way to construct a kernel matrix from a similarity matrix is: assume the similarity matrix is A. Let A=(A+A')/2. We obtain a symmetric matrix. Note that before we do the averaging the two symmetric positions, when only one of (i,j) and (j,i) is missing, we copy its symmetric counterpart to the missing value position. After doing this, for the missing values in A, we just put zeros there. If A is not a semi positive definite matrix, we add a positive constant $\lambda$ into the diagonal elements of A to make it positive definite: $K=A+\lambda I$. The $\lambda$ is set to be slightly greater than the absolute value of the minimum eigenvalue of A. We call this approach “naïve” approach. Kernel matrix constructed by this way is called “Diag” kernel. A second approach called “diffusion” kernel [2] makes use of the property of matrix exponential function. Matrix exponential always translates a symmetric matrix into a symmetric positive definite matrix. So the produced matrix can be used as kernel matrix. In this approach, $K=\exp(\beta A)$, exp is matrix exponential function. A should be a symmetric matrix. $\beta$ is a constant. The third approach, matrix exponential update [1] is an on-line algorithm. It also makes use the property of matrix exponential function, so it is closely related to the diffusion kernel. However matrix exponential update is more sophisticated. It is derived by using von Neumann divergence and square loss. Relative loss bound has been established for this
The report is organized as follows: section 2 explains matrix exponential update; section 3 discusses how it can be used for kernel learning; section 4 shows the experiment results on a drug discovery dataset; section 5 discusses the conclusion and future work.

2. Matrix exponential update

Matrix exponential update is a natural extension of the exponentiated gradient (EG) algorithm. EG’s parameter is a vector $w$ while matrix exponential update’s parameter is a symmetric positive definite matrix. At each trial $t$, the algorithm receives an instance $X_t \in \mathbb{R}^{n \times n}$ and predicts $\hat{y}_t = tr(W_tX_t)$ based on the algorithm’s current symmetric positive definite matrix $W_t$. After knowing the instance’s real label $y_t$, it incurs a loss $(\hat{y}_t - y_t)^2$ and updates $W_t$. The update’s optimization objective function is:

$$W_{t+1} = \text{argmin}_w \Delta_F(W, W_t) + \eta (tr(WX_t) - y_t)^2$$

$F$ is the Bregman divergence. Setting the derivative with respect to $W$ to zero, we have

$$\nabla F(W_{t+1}) - \nabla F(W_t) + \eta \nabla ((tr(W_{t+1}X_t) - y_t)^2) = 0$$

The problem is not solvable in closed form. A common trick is to approximate $W_{t+1}$ in the loss term by $W_t$. Then we have

$$W_{t+1} = (\nabla F)^{-1}(\nabla F(W_t) - 2\eta (tr(W_tX_t) - y_t)X_t)$$

In our case, we use von Neumann entropy as the convex function for the Bregman divergence.

$$F(W) = tr(W \log W - W)$$

$$\nabla F(W) = \log W_t, (\nabla F)^{-1}(W) = \exp(W)$$

We also add a constraint that $tr(W)=1$. The update becomes

$$W_{t+1} = \frac{1}{Z_t} \exp(\log W_t - 2\eta (tr(W_tX_t) - y_t)X_t)$$

$$Z_t = tr(\exp(\log W_t - 2\eta (tr(W_tX_t) - y_t)X_t))$$

When $W_0$ is symmetric positive definite and all the $X_t$ are symmetric, the term $(\log W_t - 2\eta (tr(W_tX_t) - y_t)X_t)$ is always symmetric so that $W_{t+1}$ is always symmetric positive definite after each update. This property is used for kernel learning.
3. Kernel learning by matrix exponential update

Section 2 gives the general case of matrix exponential update. Being specific to the case of kernel learning, each valid (not missing) element $a_{ij}$ of the similarity matrix $A_{nn}$ gives an instance pair $(X_i, y_i = a_{ij})$; $X_i$ is a all-zero matrix except at position $(i,j)$ and position $(j,i)$, it has value 0.5. $W$ is the target kernel matrix to learn; We set $W_0$ to be the $\frac{1}{n}I_{nn}$, $n$ is the dimension of $A_{nn}$. As we discussed before, since $X_i$ is always symmetric and $W_0$ is symmetric positive definite, the produced $W_{t+1}$ is always symmetric positive definite. So $W_{t+1}$ can be used as the kernel matrix.

The above algorithm is not computational efficient. Normally the similarity matrix $A_{nn}$ contains $O(n^2)$ valid elements and the computation of matrix exponential function needs $O(n^3)$. So it needs $O(n^5)$ for one pass, which is unacceptable. The way to correct this is to learn all the valid elements of $A_{nn}$ per iteration. So we have the following “batch” version of update:

$$W_{t+1} = \frac{1}{Z_t} \exp(\log W_t - 2\eta \sum_i (tr(W_t X_i) - y_i) X_i)$$

Note that the sum of loss is for all the valid elements of $A_{nn}$. We call this sum of loss “total loss”. The kernel produced this way is called “MExp” kernel.

Diffusion kernel can be seen as the first step of the “batch” version matrix exponential update with $W_0 = I_{nn}$ and appropriate learning rate. By this means, matrix exponential update is more sophisticated than diffusion kernel.

4. Experiment

(1) Data and test setup

We draw a random subsample of a drug discovery dataset CDK2. The subsample has 37 positives and 37 negatives. We build the similarity matrix by using FLEXS provided by BioSolveIt. FLEXS FlexS is a computer program for predicting ligand superpositions. For a given pair of ligands, FlexS predicts the conformation and orientation of one of the ligands relative to the other one. The produced similarity matrix is asymmetric. FlexS distinguishes between the reference ligand and the test ligand. Around 30% of elements in the similarity matrix have missing values. As explained in BioSolveIt website: “Why doesn't FlexS produce no solution at all in some cases? There is a volume overlap threshold which prevents FlexS from generating superposition solutions, where the test-ligand extends to far away from the reference-ligand.” This means that the two molecules
are so different from each other that no reasonable alignment can be made. The data is visualized in figure 1.

![Figure 1. Original similarity matrix. Brighter or warmer color means higher value.](image)

The test setup is arranged as the following: we test the three approaches introduced before: naïve approach (“Diag” kernel), diffusion kernel and matrix exponential update (“MExp” kernel). After we learn the kernel matrices, we use SVM as the kernel-learning algorithm. We also test Nearest Neighbor method. We do 10 randomly 50% (training)-50% (testing) split of the dataset. The results (test error rate) are averaged from the 10 runs.

(2) Results

a. the produced kernel matrices

We first make the original matrix symmetric as discussed in the first section. This is shown in the figure 2. (Note that in the report, the same colors in different figures don’t mean the same values. It is only relative to other colors with a figure.) All the three approaches start from this symmetric matrix.
The “naïve” approach adds a positive constant to the diagonal elements of the above matrix (figure 3).

In the following, we also show the obtained “diffusion” kernel and “MExp” kernel.
Figure 4. Diffusion kernel

Figure 5. MExp kernel
(b) Total loss of matrix exponential update

In figure 6, we show the total loss of matrix exponential update at each iteration. We can see that after 10 iterations, the total loss becomes stable.

![Figure 6: Total loss of matrix exponential update](image)

(c) Generalization error

In the following (figure 7), we show the generalization errors of SVM on the kernels produced by the three approaches. We can see matrix exponential update achieves the best test error rate. Interestingly, the test error stabilizes after 60 iterations while in the previous figure, the total loss becomes stable after 10 iterations.
The generalization error using Nearest Neighbor on the kernel matrices are shown in the figure 8. We also show the result of Nearest Neighbor on the original asymmetric similarity matrix. It achieves the best overall error rate (even considering the results using SVM). This indicates that the simple averaging method that makes the symmetric matrix may be against the nature of these data.
5. Conclusion and future work

We show in the report that matrix exponential update is a better way to learn kernel matrix from a similarity matrix. However, to make it more practical, there are still a number of issues. These need to be solved in the future.

(1) Numerical issues
Computation involving exponential function always has numerical problem. Things are even worse here since we use exp and log alternatively. So when the minimum eigenvalue of W is around zero, overflow or underflow is very easy to happen. Then the log function becomes undefined. One way to solve this is to only use exp function and avoid log function and some updates can be derived correspondingly.

(2) Large Scaled
The computation cost per iteration is $O(n^3)$. It is very expensive to work on datasets with more than a few thousand examples.

(3) Asymmetric kernel
All the kernel methods require a symmetric matrix. All the three approaches in the report have to start with a symmetric matrix. When the data is inherently asymmetric, there should be a way that systematically corrects this rather than using the simpler averaging method.

(4) Rectangular similarity matrix
We haven’t touched this problem in the project. Rectangular similarity matrix is more common in practice.

(5) Incorporate label informations into account
We know the labels of partial data (training set examples). Knowing this information maybe invaluable to kernel learning. What’s more interesting is learning kernel matrix and classifier at the same time.

References

[1] Koji Tsuda, Gunnar Raetsch and Manfred Warmuth “Matrix Exponential Updates for On-line Learning and Bregman Projections” (manuscript)
