A PARALLEL IMPLEMENTATION OF SUPPORT VECTOR MACHINES WITH NONLINEAR DIMENSIONALITY REDUCTION

SHIBO YAO, DONG WEI, WUJI LIU

1. Introduction

Support Vector Machines (SVM) [10, 3] has been widely adopted in various applications and shown surprisingly good performance. Inheriting from the perceptron methods [7], it aims to find a soft hyperplane (or multiple hyperplanes for multiclass cases) that can optimally separates different classes, and also maximizes the margin to reduce the sensitivity to noise. The linearly non-separable cases once puzzled scholars; fortunately such cases have been solved by kernel-based methods [4], which map the samples from the original space to a higher-dimensional space where they become linearly separable. The main drawback of kernel-based methods is that the optimal kernel and the best setting of the corresponding hyperparameters have to be searched, which requires a lot more time and computation and consequently this can lead to overfitting. Furthermore, from the perspective of signal processing, not all features are useful in developing models and we desire to remove the non-significant signals from the input features and condense the data representation. To solve these two problems, we use non-linear dimensionality reduction, to be more specific, Locally Linear Embedding (LLE) [8, 9], combined with SVM to improve model performance.

We implement LLE and SVM in a parallel manner such that the running time can be significantly reduced in a multicore development environment[1]. In multiclass SVM learning, there are two major strategies, namely one-vs-rest (ovr) and one-vs-one (ovo). The ovr strategy seeks n hyperplanes, each of which separates one specific class from the rest as a whole. The time complexity of ovr is linear with respect to the number of class but this usually leads to unbalanced training and unsatisfying results. The ovo strategy seeks $\frac{n(n-1)}{2}$ pairwise hyperplanes, each of which separates one specific class from another class. When making inference on a new sample, it goes through every binary classifier and follows the rule of majority vote. The time complexity of ovo is quadratic with respect to the number of class, but it generates more satisfying results compared to ovr. In LLE, one has to compute the distance matrix and perform k-nearest-neighbor search, both of which are quadratic complexity. It also involves solving least-square problems which can be embarrassingly parallelized.

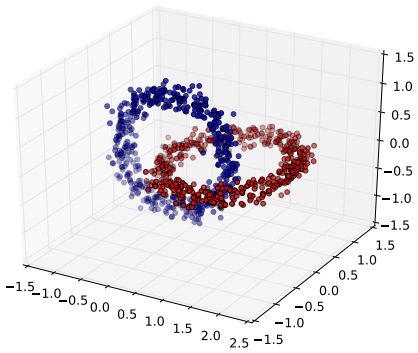


Figure 1. Twin ring, a typical Manifold

Date: April 2019, {sy372, dw277, wl87}@njit.edu.

THIS WAS A PROJECT REPORT FROM THE MACHINE LEARNING AND INFORMATION THEORY CLASS TAUGHT BY JOERG KLIEWER. WE FOUND IT MIGHT BE HELPFUL TO OTHERS AFTER SEATING IT IN DUST AND DECIDED TO PUT IT ONLINE.

[1]Github repo: `https://github.com/ShiboYao/LocallyLinearEmbedding-SVM`

2. Support Vector Machines

The optimization goal of SVM is to obtain the largest margin and meanwhile to have as few mis-classifications as possible. The optimization function [2, 1] is formed as follows,

$$\begin{aligned} \arg\min_{\mathbf{w}} \quad & \frac{1}{2}||\mathbf{w}||_2^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & \xi_i \geq 0, \\ & y_i(\mathbf{x}_i^T\mathbf{w}) \geq 1 - \xi_i \ \forall i \end{aligned}$$

where $\mathbf{w}$ is the weight vector (can include the intercept term, we always write it into linear form instead of affine), C is a hyper-parameter that controls the tradeoff between fitting the data and maximizing the margin, N denotes the number of samples for training, $\mathbf{x}_i$ is the feature vector, y_i is the corresponding label, ξ_i is the slack variable for constructing the soft margin. The loss function presented above can be shown convex, therefore if we adopt gradient-descent-like methods, the global optimum can be reached. A straightforward approach is subgradient descent. It becomes clearer if we write the optimization function into

$$argmin(\mathbf{w})\ f(\mathbf{w}) = argmin(\mathbf{w})\ \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N} max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i)).$$

If we denote the subgradient of the function above with

$$\nabla f(\mathbf{w}_0) = \lim_{\mathbf{w}\to\mathbf{w}_0} \frac{f(\mathbf{w}) - f(\mathbf{w}_0)}{\mathbf{w} - \mathbf{w}_0},$$

then

$$\nabla f(\mathbf{w}_0) = \mathbf{w}_0 - C\sum_j y_j\mathbf{x}_j, \forall j \in \{i \mid y_i(\mathbf{w}_0^T\mathbf{x}_i) < 1\}.$$

Starting with an arbitrary point $\mathbf{w}_0$ and a learning rate η, the subgradient descent process can be described as

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \cdot \nabla f(\mathbf{w}_{t-1}), \forall j \in \{i \mid y_i(\mathbf{w}_{t-1}^T\mathbf{x}_i) < 1\}.$$

Substitute the subgradient and

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta(\mathbf{w}_{t-1} - C\sum_j y_j\mathbf{x}_j) = (1-\eta)\mathbf{w}_{t-1} + \eta \cdot C\sum_j y_j\mathbf{x}_j, \forall j \in \{i \mid y_i(\mathbf{w}_{t-1}^T\mathbf{x}_i) < 1\}.$$

Note that the learning rate η should be chosen wisely and a learning rate decay is usually applied, such that a larger learning rate at the beginning can accelerate the learning process and eventually the learning rate decays to zero and the final result is as close to the global optimum as possible. And when the training data set is large, stochastic subgradient descent or mini-batch learning can be applied. The stopping criteria can be predefined based on the number of epochs or batches, however a more reliable option is to record the performance (e.g. accuracy α) on the training set and set a small tolerance ϵ such that the performance change does not go beyond the tolerance within k steps before stopping.

$$\frac{\alpha_t}{1+\epsilon} <= \alpha_{t-i}, \forall i \in \{1, ..., k\}$$

So far, we have gone through the SVM for linearly separable cases. As a matter of fact, there are much more linear non-separable cases which can be solved by kernels in SVM. To address the problems stated in the introduction section, we propose to use a non-linear dimensionality reduction approach, namely the Locally Linear Embedding (LLE), before fitting SVM. The advantages of such an approach are as follows. LLE has a closed-form solution and it has only one hyperparameter which is nonsensitive, therefore by combining LLE with linear SVM we can to a large extent simplify the whole modeling process without losing performance. LLE is able to capture the complex data distribution, which has significant superiority over linear PCA and marginalizes the usage of kernels in SVM.

In addition, to mitigate the quadratic complexity of training ovo discussed in the introduction, we implement the SVM in a parallel manner. This is merely to embarrassingly parallelize the training of $\frac{C(C-1)}{2}$ pairwise hyperplanes and algorithm parallel is by no means involved. We also parallelize the implementation of LLE to accelerate the fitting in a multi-core development environment.

3. Nonlinear Dimensionality Reduction

Lower dimensionality is desired in many machine learning tasks since it reduces the computation with more compact representation. A common practice of dimensionality reduction is Principle Component Analysis (PCA). PCA is able to stem the useful information over linear mapping in the eigen space and dispose of the noise, which can be done via Singular Value Decomposition (SVD).

Nonlinear dimensionality reduction is usually done via manipulation on the adjacency matrix or the Laplacian matrix to capture the complex data distribution. LLE assumes the data lie on a lower-dimensional manifold embedded in the original high-dimensional space. The manifold retains the Euclidean geometry properties locally and each data point is a weighted sum of its nearest neighbors. Such relationship is captured in the original space and recovered in the low-dimensional space. And therefore we obtain the low-dimensional representation. Formally, in the original m-dimensional space there are n data points and each data point is a linear combination of its k nearest neighbors, $\mathbf{x}_i = \frac{1}{k}\sum_{j=1}^{k} w_{ij}\mathbf{x}_j, i \neq j$. We want to find out all of the n weight vectors $\mathbf{w}_i$, or equally an $n \times n$ wight matrix W to represent the relationship of the data points. Thus the optimization function is

$$argmin(W)\sum_{i=1}^{n}\|\mathbf{x}_i - \sum_{j=1}^{k} w_{ij}\mathbf{x}_j\|^2$$

$$s.t.\sum_{j=1}^{k} w_{ij} = 1$$

Note that such a problem can be reduced to n least square problems with a linear equality constraint.

$$argmin(\mathbf{w}_i)\|\mathbf{x}_i - \sum_{j=1}^{k} w_{ij}\mathbf{x}_j\|^2$$

$$s.t.\sum_{j=1}^{k} w_{ij} = 1$$

To solve for the weights, we first notice that

$$\Phi(\mathbf{w}_i) = \|\mathbf{x}_i - \sum_{j=1}^{k} w_{ij}\mathbf{x}_j\|^2 = \|\sum_{j=1}^{k}\mathbf{x}_i - w_{ij}\mathbf{x}_j\|^2 = \|\sum_{j=1}^{k} w_{ij}(\mathbf{x}_i - \mathbf{x}_j)\|^2$$

If we define $\mathbf{z}_j = \mathbf{x}_i - \mathbf{x}_j$, then

$$\Phi(\mathbf{w}_i) = \|\sum_{j} w_{ij}\mathbf{z}_j\|.$$

Furthermore in matrix form, if we let $\mathbf{z} = (\mathbf{z}_1, ..., \mathbf{z}_k)$, then

$$\Phi(\mathbf{w}_i) = \mathbf{w}_i^T\mathbf{z}\mathbf{z}^T\mathbf{w}_i = \mathbf{w}_i^T G_i\mathbf{w}_i.$$

Recall the constraint of $\mathbf{w}^T\mathbf{1} = 1$. We can associate a Lagrangian multiplier λ such that (duality gap analysis omitted)

$$\mathcal{L}(\mathbf{w}_i, \lambda) = \mathbf{w}_i^T G_i\mathbf{w}_i + \lambda\mathbf{w}_i^T\mathbf{1}.$$

Taking the partial derivatives and making them equal zero, we get

$$\begin{cases} \frac{\partial\mathcal{L}}{\partial\mathbf{w}_i} = 2G_i\mathbf{w}_i - \lambda\mathbf{1} = 0 \\ \frac{\partial\mathcal{L}}{\partial\lambda} = \mathbf{w}_i^T\mathbf{1} - 1 = 0 \end{cases}$$

Therefore,

$$G_i\mathbf{w}_i = \frac{\lambda}{2}\mathbf{1}.$$

Practically, matrix G_i can be singular. To assure G_i is invertible and the numerical stability, we take a small portion δ of the trace of G_i and add it to its diagonal,

$$G_i \leftarrow G_i + \delta \cdot tr(G_i)\mathbf{I}.$$

A single weight vector would be

$$\mathbf{w}_i = \frac{\lambda}{2}G_i^{-1}\mathbf{1}$$

where we can adjust λ to ensure $\mathbf{w}_i^T\mathbf{1} = 1$. And the weight matrix based on the data representation in the original space is obtained.

The next step would be to find out the compact representations of the data points in the low-dimensional space such that they retain the relationship carried by the weight matrix W. If we let $Y \in \mathbb{R}^{d\times n}$ denote the display matrix of data points in the d-dimensional space, the optimization function would be

$$\Phi(Y) = \sum_{i=1}^{n} \|\mathbf{y}_i - \sum_j w_{ij}\mathbf{y}_j\|^2.$$

Writing it into matrix form, we get

$$\Phi(Y) = Y^T(\mathbf{I} - W)^T(\mathbf{I} - W)Y.$$

Let $M = (\mathbf{I} - W)^T(\mathbf{I} - W)$ and

$$\Phi(Y) = Y^T MY.$$

If we directly solve this optimization, we would get infinitely many optimal solutions since without the mean and covariance constraints the data points can be arbitrarily placed in the low-dimensional space. Therefore, we constrain the mean of data points to be zero and the covariance of the data points to be identity in the low-dimensional space, which means $\frac{1}{n}Y^TY = \mathbf{I}$. Again, associating a Lagrangian multiplier μ, the dual problem is

$$argmin(Y)\ \mathcal{L}(Y,\mu) = argmin(Y)\ Y^TMY - \mu(\frac{1}{n}Y^TY - \mathbf{I}).$$

Taking the derivative with respect to Y and making it zero, we get

$$\frac{\partial \mathcal{L}}{\partial Y} = 2MY - 2\frac{\mu}{n}Y = 0,$$

which is equal to

$$MY = \frac{\mu}{n}Y.$$

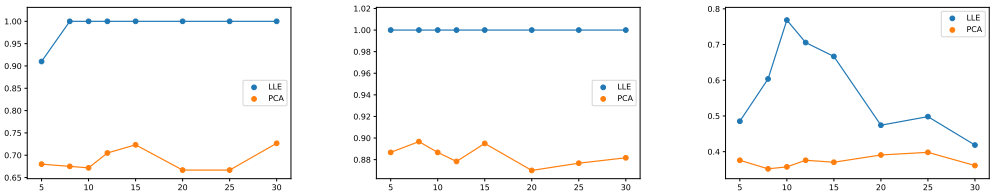
Immediately we recognize this is an eigen function. Since we are minimizing the reconstruction error, we need to do eigendecomposition on matrix M, obtain the smallest $d+1$ eigenvectors and dispose of the smallest one which is a constant eigenvector and meaningless.

4. Parallel Implementation

$$\begin{bmatrix} & w_{12} & w_{13} & \dots & w_{1C} \\ & & w_{23} & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & w_{(C-1)C} \\ & & & & \end{bmatrix} \qquad \begin{bmatrix} & d_{12} & d_{13} & \dots & d_{1n} \\ & & d_{23} & \ddots & \vdots \\ & & & \ddots & \vdots \\ & & & & d_{(n-1)n} \\ & & & & \end{bmatrix} \rightarrow \begin{bmatrix} \mathcal{N}_1 \\ \mathcal{N}_2 \\ \vdots \\ \mathcal{N}_n \end{bmatrix} \rightarrow \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

The ovo multiclass SVM involves calculation of $\frac{C(C-1)}{2}$ hyperplanes which can be displayed as the matrix on the left, where w_{ij} separates class i and class j and their associated penalization strength that balances the data fitting and mis-classifications. Since the calculations of w_{ij} do not interfere with each other and there is no information communication, we can chop the calculation into $\frac{C(C-1)}{2}$ jobs, send them to different CPU cores as distinct jobs and accelerate the whole training process. For memory efficiency, the jobs can share the data and only sample indices are passed. During the inference, a new sample goes through every binary classifier and its final prediction is the majority vote.

In LLE, we need to calculate the Euclidean distance matrix first, which can be displayed as the matrix on the right, where d_{ij} is the Euclidean distance between data point i and data point j. Since the Euclidean distance is communicative, the matrix is symmetric and we only need to compute the upper triangular part with all diagonal elements being zero. The second step of LLE is to search for k nearest neighbors for each data point. We use a partition function in quicksort to accomplish this which is $O(n)$ complex for each data point and $O(n^2)$ overall. After kNN search, we need to solve the weights via least square. The closed-form solution has been derived in the previous section and there are n reduced jobs in total. The above jobs can all be embarrassingly parallelized.



(a) Twin ring (b) Double moon (c) Handwritten digits

FIGURE 2. Sensitivity of k in LLE (accuracy in percentage)

Note that the weight matrix W is sparse due to the locality assumption. For the final step of low-dimensional coordinates reconstruction via eigendecomposition on M, we use a numerical solver provided by *scipy* [5]. We use *multiprocessing* in *python* to do parallel implementation[2].

5. Experiments

Our initial experiments are on two synthetic datasets and one real dataset. One of the synthetic datasets is presented at the beginning, which has a structure of two ring-like manifolds embedded in a 3-d space and cross each other. This type of data distribution is a big challenge to linear classifiers and even non-linear classifiers. The other synthetic dataset is the double-moons dataset. It is a relatively easier task compared to the twin-rings but still challenging to many linear classifiers.

The real dataset is the original handwritten digits dataset. The dataset contains 1797 handwritten digit images from 0 to 9. The images are 8×8 gray-scale image and rescaled to 0-1. It's a balanced dataset which means each class has approximately 180 samples. It is widely considered that images often lie on a lower-dimensional manifold embedded in the original feature space, such as handwritten digits slighted tilted or face pictures with different illumination conditions and different facial expressions. For more visualization of the datasets, please refer to our github link.

The synthetic datasets are generated with Gaussian noise ($\sigma = 0.3$) and each has 2000 samples. The optimal hyperparameter values of C which controls the tradeoff between fitting data and maximizing the margin is decided by 5-fold cross validation and grid-search. The searching space for C is [0.05, 0.1, 0.5, 1, 5, 10, 20]. As for LLE, there is one hyperparameter k which is the number of nearest neighbors. We also include sensitivity analysis on it, and sensitivity analysis on d, the dimensionality of the lower-dimensional space. Note that we perform PCA or LLE on the entire dataset which includes both training and testing, after which the SVM is trained on the training set. We always randomly select 70% as training set and the rest 30% as testing set.

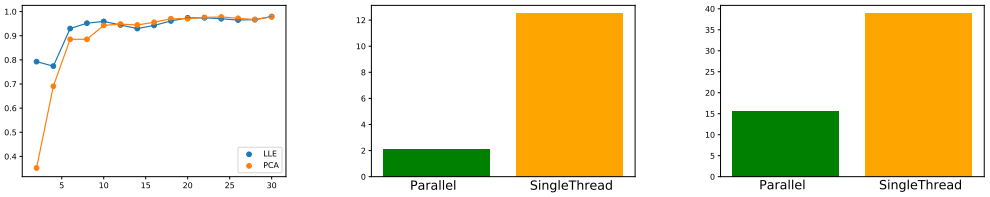
In out experiments, we first set $k = 10$ and $d = 2$. The twin-ring dataset yields a 100% accuracy on testing set with LLE and a roughly 70% accuracy (there is some randomness in data generation and train-test split) on testing set with PCA. The double-moon dataset yields a 100% accuracy on testing set with LLE and a roughly 90% accuracy on testing set with PCA. This is not surprising since we generate the synthetic datasets on purpose such that they possess manifold structure. With LLE, the digits data yields an accuracy of roughly 76% on testing set and an accuracy of roughly 37% with PCA. In the low-dimensional setting, LLE is much more capable capturing more information than PCA.

Since the two synthetic datasets are already low-dimensional, we fix d at 2 and alter k from 5 to 30 to see the sensitivity of LLE on k. Note that usually k is reasonably small such that the locality assumption is preserved, but not too small such that we do not let only a few neighbors dominate the reconstruction. We also study how the model performance varies with the dimensionality. It shows that in low-dimensional cases, LLE outperforms PCA by a large margin while in high-dimensional cases, their performance are comparable in terms of accuracy.

6. Discussion and Conclusion

In this project, we implemented multiclass-linear-SVM with LLE, a nonlinear dimensionality reduction method that is able to capture the latent manifold distribution, both of which from scratch. The efficacy of such combination is verified via synthetic and real datasets, which outperforms PCA as a linear dimensionality reduction approach. We also gained superiority in terms

[2]We implemented both LLE and linear multiclass SVM in a parallel manner and tested their correctness and effectiveness, but in the experiments we use the LLE we implemented and SVM implemented in scikit-learn which is a wrapper of more efficient C/C++ code.



(a) Sensitivity of d in LLE (b) LLE speed (c) SVM speed

FIGURE 3. handwritten digits dataset (a)Sensitivity of d in LLE (accuracy in percentage) (b)Speed Comparison of Parallel Implementation and Single Thread (execution time in seconds)

of running speed via disposing of kernel space search and parallel implementation in a multicore development environment. However, during experiments we also find that LLE is sensitive to local noise where it fails to capture the data topology and that LLE does not scale well to large dataset. Such issues can be alleviated by specifically-designed graph algorithms and approximation algorithms without significant loss of performance. We also foresee manifold learning which is originally based on geometry viewpoint will have interaction with probability and information theory, such as the powerful visualization method T-SNE [6].

REFERENCES

[1] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
[2] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*, volume 76. John Wiley & Sons, 2013.
[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.
[4] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.
[5] Eric Jones, Travis Oliphant, and Pearu Peterson. *SciPy*: Open source scientific tools for *Python*. 2014.
[6] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.
[7] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
[8] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
[9] Lawrence K Saul and Sam T Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *Journal of machine learning research*, 4(Jun):119–155, 2003.
[10] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.