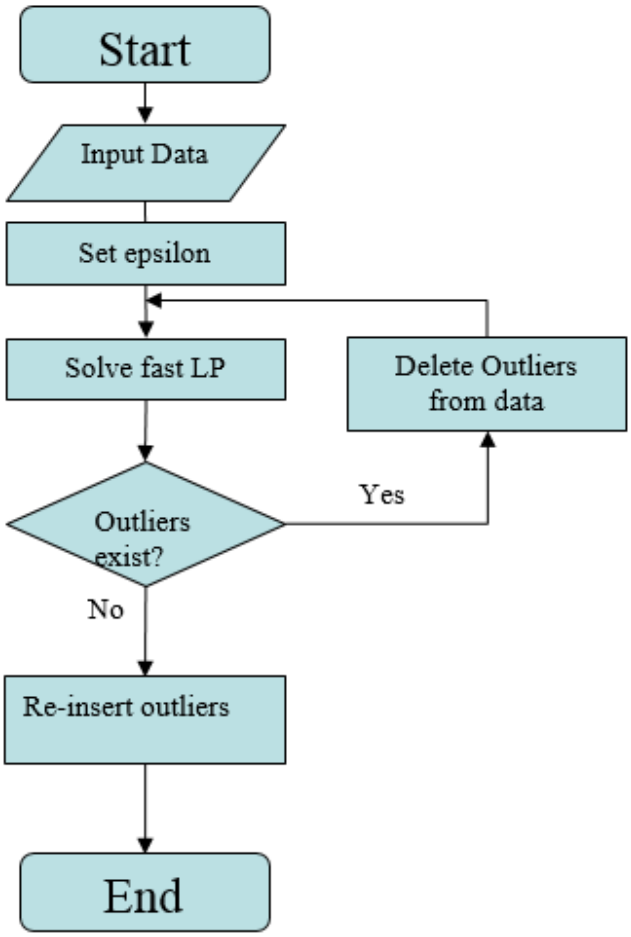
Report 10/19/2017

Our method:



What is Fast LP?

Let's see the original problem LP:

$$\max_{\mathbf{a},\mathbf{B},\mathbf{dt}} \quad \sum_{i \in \mathcal{N}_A} t_i \tag{17a}$$

$$\text{s.t.} \quad t_i + \epsilon \leq d_{ik} \qquad \forall i \in \mathcal{N}_A, k \in \bar{\mathcal{S}}_i \tag{17b}$$

$$t_i \leq d_{ij} \qquad \forall i \in \mathcal{N}_A, j \in \mathcal{S}_i \tag{17c}$$

$$t_i \geq 0 \qquad \forall i \in \mathcal{N}_A \tag{17d}$$

$$d_{ij} = \sum_{k=1}^{p} \delta_{ijk} a_k + \sum_{k=1}^{p} \delta_{ijk}^2 b_{kk} + 2 \sum_{k=1}^{p-1} \sum_{\ell=k+1}^{p} \delta_{ijk} \delta_{ij\ell} b_{k\ell} \qquad \forall i, j \in \mathcal{N} \tag{17e}$$

$$0 \leq d_{ij} \leq 1 \qquad \forall i, j \in \mathcal{N} \tag{17f}$$

$$\mathbf{a} \in \mathbb{R}^p \tag{17g}$$

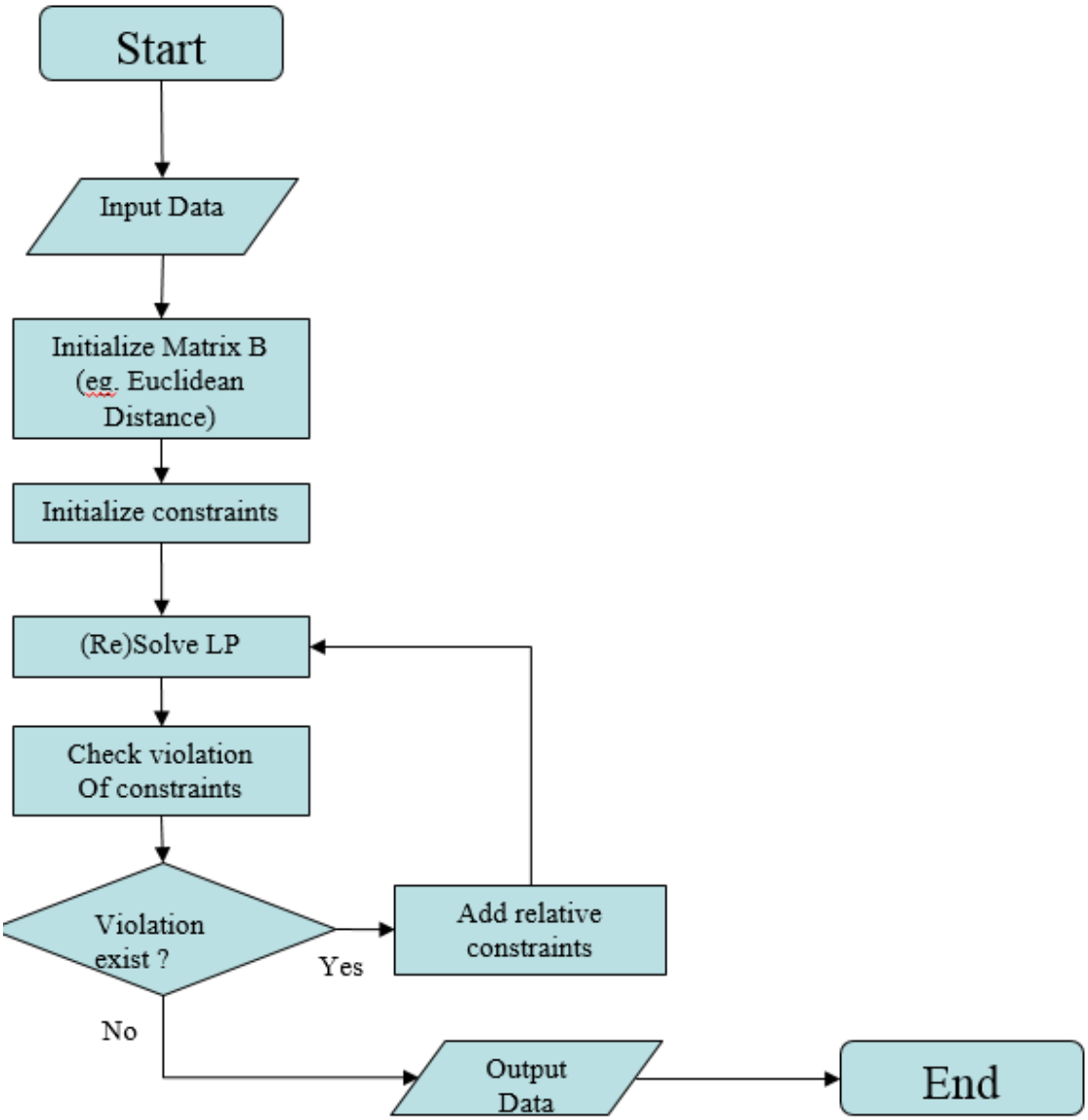
$$b_{k\ell} \in \mathbb{R} \qquad \forall k = 1, \ldots, p, \ell = k, \ldots, p \tag{17h}$$

Our goal is to train matrix B as a metric and apply it with NN method to classify data.

Fast LP is based on original LP ,they have same solution, but Fast LP runs faster.

But how? By running iterations, adding limited number of constraints (17a,17b,17c) in every iteration rather than using all constraint in 17a,17b,17c. Number of constraints can be very large if we have large dataset.

Fast LP Algorithm with Row and Column Generation:



My questions:

1. Why NN, not 2-NN, 3-NN? OK, it reduces the calculating time.

2. What if 3-NN or 6-NN performs better in accuracy? Do we need experiments on this?

3. Value of epsilon matters only when it is nearly reaching “max epsilon”, so why do we care about it ? I mean it will not change the solution if we keep epsilon less equal than “max epsilon”/2.

4. About “re-insert outliers”, we got this:

$$R_i = \frac{\min\limits_{j \in C_i} d_{ij}}{\min\limits_{k \notin C_i} d_{ik}}$$

Algorithm Improved Method of Outlier Re-insertion with Ranking of R_i';	
1:	Compute R_i for each outlier i;
2:	Identify all non-outliers j for which i is now the nearest co-class/non-class neighbor, i.e. all j whose R_i values are affected by insertion of i;
3:	Define $R_i' = \min(R_i, \min\limits_j R_j)$;
4:	Sort outliers in the order of decreasing R_i';
5:	Insert the outliers according to decreasing ranking of R_i';
6:	Repeat steps 1-5 until no further re-insertion is possible, i.e. remaining outliers all have R_i'<1

If Metric does not change after we put points back, so why do we do that? It's meaningless.

5. About what to do next?

(1) See how initialization of Matrix B influence the solving time in Fast LP.

(2) Comparison of our method and others(k-nn ,svm…) with different dataset.

(3) Improving of the method (hardest)