1 Introduction

A fundamental question of machine learning is how to compare examples. If an algorithm could perfectly determine whether two examples were semantically similar or dissimilar, most subsequent machine learning tasks would become trivial (i.e., the 1-nearest-neighbor classifier will achieve perfect results). A common choice for a dissimilarity measurement is an uninformed norm, like the Euclidean distance. While simple, it does not necessarily represent the similarity in the problem’s domain.

Metric learning algorithms, such as NCA [8], ITML [5] and LMNN [11] address this issue and are tuned for learning a Mahalanobis similarity. These methods learn a linear transformation where the Euclidean distance in the transformed space reflects the actual similarity between the data instances as defined in the problem domain (such as similarity in class labels).

While representing the data in the Mahalanobis space can improve classification accuracy, applying Metric learning can also reduce its dimensionality. By restricting the learned transformation matrix to be rectangular, we can generate a lower dimensional representation of high dimensional data. Experimental results [12] show that such a simple linear transformation can not only significantly improve the performance of kNN, but also substantially decrease the amount of testing time: Classifying test data using kNN can be achieved faster on the lower dimensional representation using dedicated data structures such as KD-Trees [1] or Ball Trees [3], which speed-up significantly in low dimensional settings.

In this writeup, we focus on Large Margin Nearest Neighbors (LMNN) [11]. LMNN employs a convex optimization that guarantees a global optimal solution. In LMNN, for every data instance \( x \) we choose \( k \) dedicated target neighbors, which we would like to be the actual nearest neighbors after applying the transformation (marked as \( j \rightarrow i \) if \( x_j \) is a target neighbor for \( x_i \)). A common heuristic for choosing target neighbors for an instance is picking the closest instances with the same label. LMNN has two objectives: First, it pulls target neighbors together, reducing the distance between them. Second, for different labeled instances (impostors), it moves them further away so the distances to impostors should exceed the distances to target neighbors by a large margin. A constant \( \mu \) defines the trade-off between the objectives. Formally, the loss function can be represented as:

\[
\mathcal{L}(L) = \sum_{j \rightarrow i} \left( \frac{\|L(x_i - x_j)\|^2_2}{\text{pull target neighbors}} + \mu \sum_{k \text{ s.t. } y_i \neq y_k} \max(0, 1 + \|L(x_i - x_j)\|^2_2 - \|L(x_i - x_k)\|^2_2) \right)
\]

There have been previous efforts to extend LMNN beyond the Mahalanobis by kernelization [4, 7, 10]. While these methods do show improvement for several datasets, they are inherently limited by the size of the data \( n \), as it requires the construction of a kernel matrix of size \( n \times n \). In contrast, we
expand LMNN to go beyond the Mahalanobis distance by substituting the linear feature mapping with non-linear Gradient Boosting Regression Trees [2] (GBRT).

2 Building non linear mapping for metric learning

Let \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{R}^d \times C \) be all training data, with \( C = \{1, \ldots, c\} \). LMNN learns the mapping \( x \to Lx \), where \( L \in \mathbb{R}^{r \times d} \). We extend LMNN towards a more general non-linear mapping \( x \to \phi(x) \) and define the loss function with accordance to \( \phi(x) \):

\[
L(\phi) = \sum_{j \to i} (\|\phi(x_i) - \phi(x_j)\|_2^2 + \mu \sum_{k \text{ s.t. } y_i \neq y_k} \max(0, 1 + \|\phi(x_i) - \phi(x_j)\|_2^2 - \|\phi(x_i) - \phi(x_k)\|_2^2)) \quad (2)
\]

Here, we define \( \phi(x) = [H_1(x), \ldots, H_r(x)]^\top \), where \( H_q(x) \) is learned with GBRT. Gradient boosting [2] learns an ensemble \( H_q = \sum_{t=1}^T \alpha_t h^q_t \), where each \( h^q_t \) is a regression tree with limited depth, built with CART [6,9]. The learning rate for iteration \( t \) is denoted by \( \alpha_t \). Each tree \( h^q_t \) approximates the gradient at iteration \( t \) with respect to dimension \( q \):

\[
h^q_t \approx \left[ \frac{\partial L(\phi)}{\partial \phi(x_1)_q}, \ldots, \frac{\partial L(\phi)}{\partial \phi(x_n)_q} \right]^\top \quad (3)
\]

As the learned non-linear function is non-convex, initialization affects the optimization result. We initialize with the linear transformation learned by LMNN. By defining \( L_{q,:} \) as the \( q \)th row in \( L \), the final non-linear mapping for a datapoint (labeled or unlabeled) becomes \( \phi(x) = [H_1(x), \ldots, H_r(x)]^\top \) where:

\[
H_q(x) = L_{q,:}x + \sum_{t=1}^T \alpha_t h^q_t(x) \quad (4)
\]

3 Results

We include two sets of experiments detailed in the tables below. The first set compares the error of \( k \)-nearest-neighbor classification with the LMNN metric and that of our non-linear mapping (labeled as GB-LMNN). The second set of experiments take a different set of datasets of smaller size \( n \) and compares with Kernelized LMNN [10] (K-LMNN). K-LMNN is not compared on all datasets due to extensive computational requirements caused by learning a matrix of size \( n \times n \). For these experiments we used \( k = 3, \mu = 1 \) and a validation set of 15% of the training set (20% – 25% on the smaller datasets). Smaller datasets testing sets were generated by 10 splits of the data and cross-validation. Emphasized in bold are the best results for each setting.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dims</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digits</td>
<td></td>
<td>6.12</td>
<td>5.33</td>
<td>5.23</td>
<td>5.18</td>
<td>6.79</td>
<td>4.94</td>
<td>3.91</td>
<td>4.17</td>
</tr>
<tr>
<td>Isolet</td>
<td></td>
<td>3.58</td>
<td>2.1</td>
<td>1.87</td>
<td>1.65</td>
<td>3.58</td>
<td>2.1</td>
<td>1.87</td>
<td>1.65</td>
</tr>
<tr>
<td>Mnist</td>
<td></td>
<td>5.13</td>
<td>4.63</td>
<td>4.53</td>
<td>4.24</td>
<td>5.9</td>
<td>4.62</td>
<td>4.04</td>
<td>3.78</td>
</tr>
<tr>
<td>Letters</td>
<td>All (16)</td>
<td>3.47</td>
<td>6.12</td>
<td>2.88</td>
<td>5.27</td>
<td>5.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bal</td>
<td>All (4)</td>
<td>6.7</td>
<td>3.33</td>
<td>5.8</td>
<td>5.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-LMNN</td>
<td>N/A</td>
<td>2.03</td>
<td>6.63</td>
<td>2.05</td>
<td>4.62</td>
<td>4.67</td>
<td>4.79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dims</th>
<th>Letters</th>
<th>All (16)</th>
<th>Bal</th>
<th>All (4)</th>
<th>Iris</th>
<th>All (4)</th>
<th>Ion</th>
<th>All (34)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LMNN</td>
<td></td>
<td>3.47</td>
<td>6.12</td>
<td>2.88</td>
<td>5.27</td>
<td>5.27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-LMNN</td>
<td>N/A</td>
<td>6.7</td>
<td>3.33</td>
<td>5.8</td>
<td>5.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GB-LMNN</td>
<td></td>
<td>2.03</td>
<td>6.63</td>
<td>2.05</td>
<td>4.62</td>
<td>4.67</td>
<td>4.79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References


