

Is that you?

Metric Learning Approaches for Face Identification

Debtanu Gupta (2019701003)

Jeet Vora(2019701006)

Tushar Chandra(2019701014)

Github link : https://github.com/debtanu177/face_identification.git



Overview

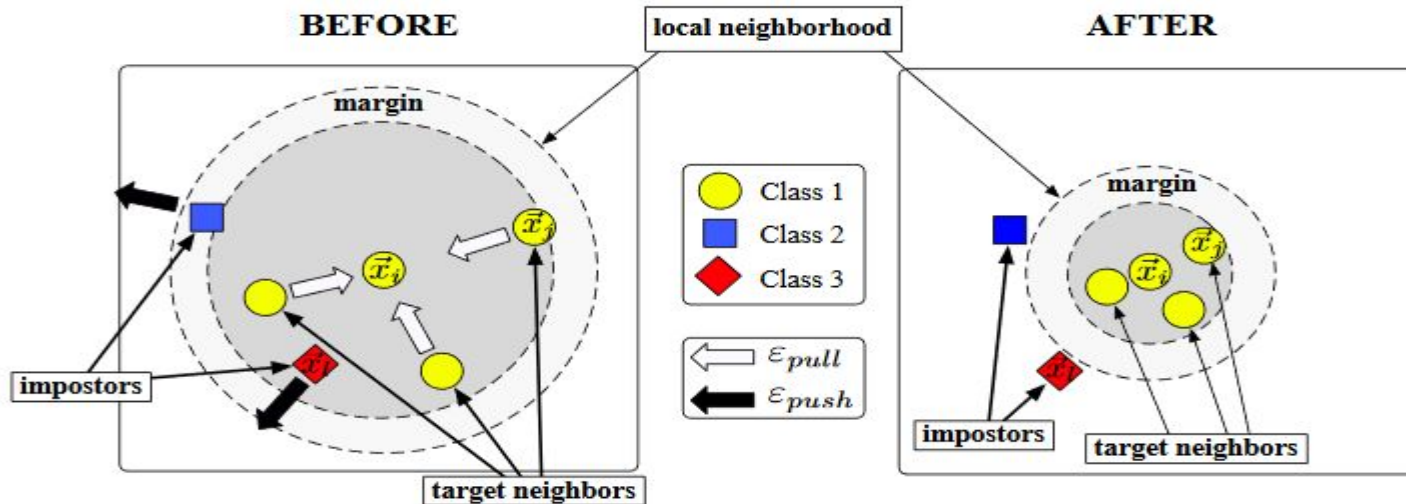
- Brief description of the project
- LMNN
- MKNN
- Implementation of LMNN
- Implementation of MKNN
- Results & Observations

Brief description of the project


- In this paper they proposed two methods for face identification based on learning Mahalanobis metrics over a given representation space.
 - **LDML:** uses logistic discriminant to learn a metric from a set of labelled image pairs. Its objective is to find a metric such that positive pairs have smaller distances than negative pairs.
 - **MkNN:** uses a set of labelled images, and is based on marginalising a k-nearest-neighbour (kNN) classifier for both images of a pair. The MkNN classifier computes the marginal probability that the two faces are the same person, i.e. marginalising over who that exactly is.
- In this Project we are implementing MkNN method of face identification on unrestricted setting of LFW dataset.
- **Marginalized kNN (MkNN)** binary classifier for a pair of images (x^i, x^j) is based on how many positive neighbour pairs we can form from neighbours of x^i and x^j .
- **LMNN** base metric is used to define the neighbours.
- We have implemented LMNN based method so far and implemented KNN using that metric on unrestricted data.

LMNN (Large margin Nearest Neighbour)

- Maintaining a large (finite) distance between impostors and the perimeters established by target neighbors. This robustness criterion gives rise to the name of approach : large margin nearest neighbor(LMNN) classification
-



LMNN (Large margin Nearest Neighbour)


$$\mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M}(\vec{x}_i - \vec{x}_j)$$

$$\mathbf{M} = \mathbf{L}^\top \mathbf{L}.$$

$$D_{\mathbf{L}}(\vec{x}_i, \vec{x}_j) = \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|_2^2$$

- The first term in the loss function penalizes large distances between each input and its target neighbors. In terms of the linear transformation of the input space, the sum of these squared distances is given by: $\epsilon_{\text{pull}}(\mathbf{L}) = \sum_{j \rightsquigarrow i} \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2.$

The gradient of this term generates a pulling force that attracts target neighbors in the linearly transformed input space

LMNN (Large margin Nearest Neighbour)

- The second term in the loss function penalizes small distances between differently labeled examples. To simplify notation, we introduce a new indicator variable $y_{il}=1$ if and only if $y_i=y_l$, and $y_{il}=0$ otherwise. In terms of this notation, the second term of the loss function ϵ_{push} is given by:

$$\epsilon_{push}(\mathbf{L}) = \sum_{i,j \rightsquigarrow i} \sum_l (1 - y_{il}) \left[1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2 \right]_+$$

generates a pushing force that repels imposters away from the perimeter established by each example's knearest (similarly labeled) neighbors

- Finally, we combine the two terms $\epsilon_{pull}(\mathbf{L})$ and $\epsilon_{push}(\mathbf{L})$ into a single loss function for distance metric learning. The two terms can have competing effects—to attract target neighbors on one hand, to repel impostors on the other. A weighting parameter $\mu \in [0,1]$ balances these goals:

$$\epsilon(\mathbf{L}) = (1 - \mu) \epsilon_{pull}(\mathbf{L}) + \mu \epsilon_{push}(\mathbf{L}).$$

LMNN (Large margin Nearest Neighbour)

- We have, $M=L^TL$. With this change of variable, we can rewrite the squared distances that appear in the loss function. Recall that $D_M(x_i, x_j)$ denotes the squared distance with respect to the Mahalanobis metric M . This distance is equivalent to the Euclidean distance after the mapping $x_i \rightarrow Lx_i$. As a result, we obtain the loss function as,

$$\varepsilon(\mathbf{M}) = (1 - \mu) \sum_{i, j \rightsquigarrow i} \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) + \mu \sum_{i, j \rightsquigarrow i} \sum_l (1 - y_{il}) [1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_l)]_+$$

- The gradient computation can be done most efficiently by careful book-keeping from one iteration to the next. As simplifying notation, let $C_{ij} = (x_i - x_j)(x_i - x_j)^T$. It is straightforward to express the distances, in terms of this notation. In particular, at the t^{th} iteration, we have $D_t(x_i, x_j) = \text{tr}(\mathbf{M}_t C_{ij})$. Consequently, we can rewrite the loss function as,

$$\varepsilon(\mathbf{M}_t) = (1 - \mu) \sum_{i, j \rightsquigarrow i} \text{tr}(\mathbf{M}_t \mathbf{C}_{ij}) + \mu \sum_{j \rightsquigarrow i, l} (1 - y_{il}) [1 + \text{tr}(\mathbf{M}_t \mathbf{C}_{ij}) - \text{tr}(\mathbf{M}_t \mathbf{C}_{il})]_+$$

LMNN (Large margin Nearest Neighbour)

- The above equation is a piecewise linear with respect to M_t . Let us define a set of triples N_t , such that $(i,j,l) \in N_t$ if and only if the indices (i,j,l) trigger the hinge loss in the second part of Eq.. With this definition, we can write the gradient G_t of $\varepsilon(M_t)$ as,

$$\mathbf{G}_t = \frac{\partial \varepsilon}{\mathbf{M}_t} = (1 - \mu) \sum_{i,j \rightsquigarrow i} \mathbf{C}_{ij} + \mu \sum_{(i,j,l) \in \mathcal{N}^t} (\mathbf{C}_{ij} - \mathbf{C}_{il}).$$

- We can use this fact to derive an extremely efficient update that relates the gradient G_{t+1} at iteration $t+1$ from the gradient G_t at iteration t . The update simply subtracts the contributions from triples that are no longer active and adds the contributions of those that just became active:

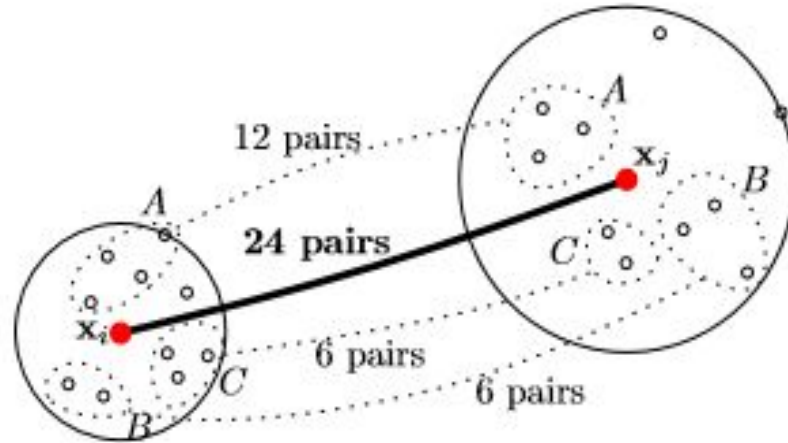
$$\mathbf{G}_{t+1} = \mathbf{G}_t - \mu \sum_{(i,j,l) \in \mathcal{N}_t - \mathcal{N}_{t+1}} (\mathbf{C}_{ij} - \mathbf{C}_{il}) + \mu \sum_{(i,j,l) \in \mathcal{N}_{t+1} - \mathcal{N}_t} (\mathbf{C}_{ij} - \mathbf{C}_{il}).$$

MkNN (Marginalised K Nearest Neighbour)

- The probability of class c for x_i is $p(y^i = c|x^i) = n_c^i/k$, where n_c^i is the number of neighbours of x^i of class c . Here, we have to predict whether a pair of images (x^i, x^j) belongs to the same class, regardless of which class that is, and even if the class is not represented in the training data. We compute the marginal probability that we assign x^i and x^j to the same class using a kNN classifier, which equals.

$$\begin{aligned} p(y_i = y_j | \mathbf{x}_i, \mathbf{x}_j) &= \sum_c p(y_i = c | \mathbf{x}_i) p(y_j = c | \mathbf{x}_j) \\ &= k^{-2} \sum_c n_c^i n_c^j. \end{aligned}$$

MkNN (Marginalised K Nearest Neighbour)



- Schematic representation of $k = 10$ neighbourhoods for x_i and x_j , and the 24 neighbour pairs (out of 100) that have the same name and contribute to the score.

Implementation

- **Dataset:** Labeled Face in the Wild (LFW) dataset.
- **Data Preprocessing:** We removed all the labels having less than 5 examples, because we are implementing KNN with 5 neighbours. After preprocessing we are left with 318 labels and 6084 samples.
- **Data Normalization:** Normalized the data
- **Dimensionality reduction:** Reduced dimensionality using PCA with 300 n_components.
- **Applied LMNN and MkNN:** Performed LMNN to find out L matrix and then transformed the original data to LMNN metric space and applied MkNN, to check if implementing LMNN actually improves the performance.



Results

- We had carried out our experiments on a smaller subset of the dataset.
- Before applying LMNN we had applied KNN using Euclidean Distance of the original space with value of $K = 5$ and we got Accuracy of 22.36% over 318 different classes.
- Then we applied LMNN and transformed into LMNN metric space and got an Accuracy of 57.89% using MkNN with value of $K = 5$, which was 55.76% before using LMNN.



Observations

- Accuracy in this process is not too good, one reason for that could be that not all classes have same no of samples. So we did an experiment where we had considered only those classes which have more than 30 samples. In this case the accuracy after LMNN was nearly 61%. So having more no of training data could help the overall training process.
- We have used 5 neighbours through all the experiments, increasing the no of neighbour doesn't really help
- Initially we were using 200 principal components in PCA. But when we increased it to 300 accuracy increased. So we should do more experiments on PCA to find out more accurate no of principal components that would increase accuracy.