EE4-68: Pattern Recognition - Coursework 2

Jorge Sanmiguel - CID: 01064565 Ricardo Vera - CID: 01061547
Electrical and Electronic Engineering Department
Imperial College London

1. Summary

The problem of re-identification aims at recognising a person from different cameras. Using the feature vectors extracted from training the ResNet50 with triple loss on CUHK03 images, the aim of this report is to examine some metric learning methods that will separate the data optimally by classes, in an aim to solve the person re-identification problem. We examine the performance of the methods such as euclidean nearest neighbours and k-means to establish a baseline to which compare, and then propose and compare a set of transformations and optimisation algorithms which learn the best data separating metric.

2. Problem Formulation

Person re-identification is to be achieved in this assignment; given a picture of a person from a point of view, that same person is to be identified from a different point of view (e.g. isolating the same person in two different camera streams). The objective is to design an algorithm that returns a set of possible matches from a set of images given a query image, with the largest accuracy possible.

This report will focus on distance metric learning. A good distance metric will separate data in an optimal way: will bring same class images close together and different class images apart in the feature space. Common distance metrics will be explored as a baseline first, and then more advanced approaches explored. From the Mahalanobis distance, Equation 2, it can be derived, using the Cholesky transform (for semi-positive matrices), that a learned distance metric is equivalent to transforming the data to a different space, followed by using the Euclidean distance in that space. The objective is to learn such transformation that yields the best data separtion.

Nearest neighbours is the algorithm of choice to obtain rank lists for query images. To evaluate the correctness of the distance metrics, rank-k accuracies together with the mean average precission (mAP) [1] will be reported. The former returns the accuracy of the correct label being present within the k nearest neighbours, and the latter takes into account the order in which the correct matches appear on the rank lists.

$$\text{Loss Function} = \sum_{i,j} y_{ij} d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) \tag{1}$$

$$\begin{aligned} d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) &= (\mathbf{x_i} - \mathbf{x_j})^T \mathbf{A} (\mathbf{x_i} - \mathbf{x_j}) \\ &= (\mathbf{G}\mathbf{x_i} - \mathbf{G}\mathbf{x_j})^T (\mathbf{G}\mathbf{x_i} - \mathbf{G}\mathbf{x_j}) \end{aligned} \tag{2}$$

Equation 1 shows the general loss function to be minimised, where $y_{ij} = 1$ if the labels of $\mathbf{x_i}$ and $\mathbf{x_j}$ are the same, and -1 if otherwise. This way, distance between similar images will increase the loss function, whilst distances between dissimilar images will decrease it. Minimising this loss function will result in reducing the distance between similar images and increasing the distance between dissimilar ones. The loss function will be further developed in Section 5.

3. Dataset

The dataset used is derived from CUHK03, a very popular dataset containing 13,164 images of 1,360 pedestrians from 2 different cameras. Feature vectors of length 2048 (one per image) have been beforehand extracted from the images, and provided to us for the distance learning problem. They are provided already partitioned into a training, gallery and query sets, together with information on the label of each feature vector and camera id.

The training set is to be used to learn the distance metric, which can be partitioned to form a validation set. The validation set has been made to contain all feature vectors corresponding to approximately 100 different individuals. The gallery and query sets are used for testing the learned model performance: each feature vector from the query set is checked against the gallery set, and the k-nearest neighbours returned to form the rank list. Note that in order to perform person re-identification in the testing phase, the gallery images with same label and camera id as the query images have to be excluded.

4. Baseline

Two baseline approaches are proposed to use as a comparison metric on the performance of the different learning approaches proposed. Their implementation is done directly on the feature vectors given on the dataset, without any distance metric transformation.

4.1. k-Nearest Neighbours

k-Nearest Neighbours is the first baseline studied. A model is trained on the gallery set provided, and for each query image, the closest k neighbours are extracted. For efficiency, instead of fitting the model multiple times to account for the cameras ids, the model was fitted only once, and then the negative matches in the neighbours list removed. Leaving the images of the same camera but of different label results in an increased challenge to the baseline model, due to the presence of a larger gallery set to which pick neighbours from. A visual representation is shown in Figure 3 (Appendix II).

Rank-k accuracies can be seen in Figure 1 and Table 1. Three common distance metrics are compared, Euclidean,

Manhattan and Cosine. All measures give similar results, with Cosine giving marginally better results. There is a sharp increase on accuracy starting from $\sim 47\%$ for Rank 1, and stabilises at $\sim 95\%$ for Rank 100 onwards. The mean average precision (mAP) is around 47% for all three metrics, with Cosine obtaining the best value of 47.82%.

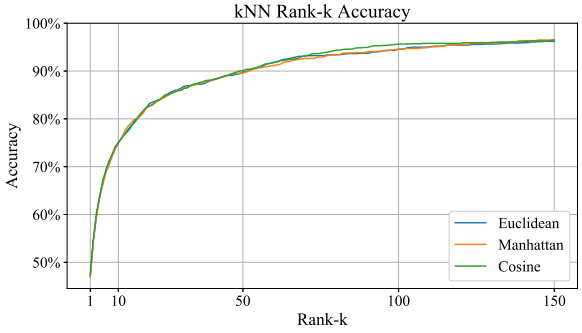


Figure 1: Rank-k accuracy for k-Nearest Neighbours using three different distance metrics: Euclidean, Manhattan and Cosine

Distance Metric	Rank 1	Rank 10	mAP
Euclidean	47.00 %	74.93 %	47.26 %
Manhattan	47.21 %	75.07 %	47.44 %
Cosine	47.57 %	75.07 %	47.82 %

Table 1: Rank 1, 10 and mAP accuracies for kNN with Euclidean, Manhattan and Cosine distance metrics.

Given that the three metrics explored give similar results, Euclidean will be used as the baseline for future comparisons, due to it being the most common and straight forward method. Furthermore, the proposed improved approaches that rely on distance metric learning work by learning a mapping, transforming features and then using Euclidean distances for kNN, hence allowing for a nice metric on the correctness of the distance metric learned.

4.2. k-Means Clustering

Using k-Means for clustering followed by kNN was also considered. Unsupervised clustering is first performed to cluster similar feature vectors. Then, for each query image the closest cluster centre is found, and the feature vectors from that cluster are taken to be a sub-gallery set on which kNN is performed to form the rank list of the query image.

Once the clustering is learned, performing k-Nearest Neighbours is much faster, due to the reduced set of feature vectors that have to be compared. kNN should also result in less mismatches on the rank lists, since the feature vectors corresponding to other classes would have been excluded by the clustering algorithm.

In theory, this formulation sounds solid, however the assumption is that the clusters will be learned such that each encompasses a class/set of classes. In practice, this is not the case, and this method implementation showed disappointing results. Using two clusters resulted in the rank-k accuracies falling by $\sim$50% (rank-1: 27.93%, rank-10: 48.64%), with this accuracy loss being even more pronounced as the number of clusters increases. See Figure 4 in Appendix II. Due to this results, kMeans will not be used as a baseline.

5. Improved Approach

Several improved approaches are proposed, and their performance evaluated.

5.1. Methods Descriptions

Kernel Based Method

Using a kernel is proposed to map the feature vectors to a higher dimensional space, hence allowing linear separators to yield better accuracies than normally possible. In the case of Nearest Neighbours, using a kernel can help rearrange the features in the space in such a way for the mean average precision to increase.

The kernel can be seen to be equivalent of the distance operation, hence the euclidean distance is substituted by a kernel, after which normal nearest neighbours comparison is performed to obtain the rank-k accuracies. A group of kernels will be tested and compared with the baseline.

Squared Mahalanobis Distance

Another method proposed is using the *Squared Mahalanobis distance* [2], which measures the distance between two points taking into account the scale of the features. This implies transforming the data with the inverse co-variance matrix of the training data. Equation 3 shows the loss function for this metric where Σ represents the co-variance matrix.

$$d_M(\mathbf{x_1}, \mathbf{x_2}) = (\mathbf{x_1} - \mathbf{x_2})^T \Sigma^{-1} (\mathbf{x_1} - \mathbf{x_2}) \tag{3}$$

This distance has the effect of normalising the features of the images. It prevents scaling effects from affecting the Nearest Neighbors performance, such as one feature outweighing the other features, or all elements of a class being spread with a large variance across the feature space.

Large Margin Nearest Neighbour

The *Large Margin Nearest Neighbour (LMNN)* algorithm, as first introduced by Weinberger et al. [3], is a metric learning approach to solve problems such as the one discussed in this report. A transformation which results in all classes to be separated by a margin of size 1 is searched. Given a set of triplets S, the objective function to minimise is:

$$\begin{aligned} \min_{\mathbf{A}} \textstyle\sum_{(x_i,x_j)\in S} d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) + \gamma \textstyle\sum_{(x_i,x_j,x_k)\in R} \xi_{ijk} \\ \text{s.t. } d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_k}) - d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_l}) \geq 1 - \xi_{ijk} \\ \xi_{ijl} \geq 0, \mathbf{A} \succ 0 \end{aligned} \tag{4}$$

Where the slack variable ξ_{ijk} relaxes the strict condition of the margin, and allows the introduction violations of the margin. The third constraint makes sure the matrix $\mathbf{A}$ is positive definite, making sure negative distances are not possible, also allowing the *Cholesky* decomposition to be performed and hence being able to project the data to the desired space with a simple matrix multiplication.

The LMNN library used uses optimisation techniques to find the matrix $\mathbf{A}$ to fulfil the requirements, due to the lack of a closed form solution.

LMNN variant with a self proposed optimiser

The final proposed method is a self-implemented optimiser to minimise the objective function in Equation 4. The optimiser is based on stochastic gradient descent and the validation set is used to evaluate the cost on every epoch iteration. The main objective is as above, hence an $\mathbf{A}$ is searched that will cluster classes the most.

The issue with this approach is the non-trivial derivation of the gradient of the loss function. Additionally, as with all optimisation problems, the possibility of reaching a local minima is very likely, as with 2048 different features it is unlikely the scenario of a convex function.

To perform any gradient method, it is first required to compute the gradient of the loss function. Since the matrix $\mathbf{A}$ is only present in the distance calculation of the loss function, the derivative of the distance with respect to matrix $\mathbf{A}$ will be derived. For the full derivation see Appendix I.

The notation used is the following: letting $\mathbf{y} = (\mathbf{x_i} - \mathbf{x_j})$ and $\mathbf{z} = \mathbf{A}\mathbf{y}$, the Mahalanobis distance in Equation 1 can be rewritten as $d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{y}^T\mathbf{A}\mathbf{y} = \mathbf{y}^T\mathbf{z}$.

The derivative of the distance $d_{\mathbf{A}}$ with respect to $\mathbf{A}$ can be split via the product rule to express it as in Equation 5, where $\mathbf{z_k}$ is the k^{th} element of the vector $\mathbf{z}$.

$$\begin{aligned}\frac{\delta(d_{\mathbf{A}})}{\delta \mathbf{A_{ij}}} &= \frac{\delta \mathbf{y^T z}}{\delta \mathbf{A_{ij}}} = \frac{\delta \mathbf{y^T z}}{\delta \mathbf{z_k}} \frac{\delta \mathbf{z_k}}{\delta \mathbf{A_{ij}}} \\ &= \mathbf{y_k}\delta_{ki}\mathbf{y_j} = \mathbf{y_i}\mathbf{y_j}\end{aligned} \tag{5}$$

Equation 5 can then be rewritten as $\frac{\delta(d_{\mathbf{A}})}{\delta \mathbf{A}} = \mathbf{y}\mathbf{y}^\mathbf{T}$, avoiding element-by-element computations. This derivative is the outer product of the distance vector between the points, and will be used for gradient descent.

Two losses for gradient descent are proposed. The first is given in Equation 6, which deals with maximising the distance between dissimilar images and minimising the distance between similar images, as discussed in Section 2.

$$\text{Loss} = \sum_{i,j} y_{ij} d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) \tag{6}$$

where $y_{ij} = 1$ if labels are the same and -1 if they are different. The loss function depends on the distance and the label similarity to separate the classes as much as possible, and is computed for all possible combination of points.

The second method considered is a variation of the first, but including a margin between similar and dissimilar points. The method implies the use of triplets as inspired by Alexander Hermans et al. in their paper for triplet loss defence [4]. The loss function in this case is given by Equation 7.

$$\text{Loss} = \sum_{\substack{i,j,k \\ y_i = y_j \neq y_k}} \Big[1 + d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) - d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_k})\Big]_+ \tag{7}$$

The margin is set to 1, and $\max(\text{Cost}(i,j,k), 0)$ is used to encourage the optimisation algorithm to focus on the set of triplets that violate the margin constraint. To compute the loss, triplets of points are needed, an anchor, a similar point, and a dissimilar one. Hence, this set of points is calculated before starting the optimisation process, and is composed of the most difficult set of points (that is picking the similar point that is furthest away from the anchor, and the dissimilar point that is closest), but also from a set of easier, randomly picked, to prevent overfitting to outliers.

5.2. Implementation

Kernel Based Optimiser

Various kernels were explored when trying to separate the data. By making use of the kernel the data is transformed, and the output of the kernel is used as the distance between the samples. The subsequent operations regarding nearest neighbours are then performed.

$$\kappa_{\mathbf{A}}(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma \sum_i \frac{(\mathbf{x_i} - \mathbf{y_i})^2}{\mathbf{x_i} + \mathbf{y_i}}\right) \tag{8}$$

See Figure 2a for rank-k accuracies. The Chi squared kernel, Equation 8, returns the best results, although they are only marginally better than the baseline. The Gaussian kernel yields the exact same accuracy as the baseline. This makes sense, as it does not alter the relative position between the elements, it only scales them. Hence kNN will find the same neighbours as in the baseline. The Laplacian kernel yields slight worse results than the baseline.

Overall, kernels do not help much in achieving higher ranklist accuracies. However this is expected, as using the standard available kernels does not involve learning of any sort, and thus they do not adapt to the data that is required to be clustered.

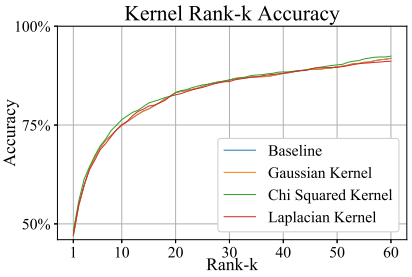
Squared Original Mahalanobis

The Squared Mahalanobis metric implementation is straightforward, the covariance matrix is *'learnt'* from the training set and then its inverse is used to transform the gallery and query features. Last, kNN is performed to obtain the ranklists. The training is a one step process that has closed form solution. There is no way of optimising the matrix, and hence the validation dataset is not used.

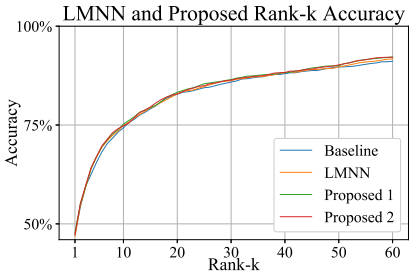
This method underperforms strongly in comparison with the baseline, causing the accuracies to fall by approximately a third (Rank-1: 30.79 %, rank-10: 49.29 %, mAP: 24.81%), see Figure 5 in Appendix II. Our impression is that Mahalanobis, when normalising the features, actually brings them together, as it does not take into account the features labels when learning.

Large Margin Nearest Neighbour

The LMNN algorithm is implemented using the library *metric-learn* [5], which perform optimisation starting from the identity matrix. The LMNN algorithm is run for two varying parameters, the regulariser γ, and the number of points considered at each time, k. Once trained, the mAP for each $\mathbf{A}$ learned with the feature vectors of the validation set, which have been randomly split into a gallery and query set, is computed. Results shown in Table 2c. Note that these mAP results are much higher than those obtained from the test set, which gives the impression that the test set is more difficult to learn than the training set.



(a) Kernel method



(b) LMNN and custom methods

k	Regulariser, γ 0.1	0.01	0.001	0.0001
3	74.22%	74.23%	74.23%	74.35%
5	74.06%	73.92%	73.92%	73.78%

(c) Validation mAP for LMNN

Method	Rank 1	Rank 10	mAP
LMNN	46.86%	74.93%	47.27%
Proposed 1	47.86%	75.21%	47.57%
Proposed 2	47.36%	74.79%	47.48%

(d) LMNN and custom test accuracies

Figure 2: Rank-k accuracies for the improved approaches. Larger figures available in Appendix III

Figure 2b shows the rank-k accuracy when using the transformation computed for the parameters $\gamma = 0.0001$ and $k = 3$. It can be seen that the resulting transformation does not improve significantly performance and has similar results to the baseline method. This might be caused by starting optimisation from the identity matrix $\mathbf{I}$; the optimisation algorithm rapidly falls into a local minima causing this result.

Self proposed optimiser

Having a custom implementation of a gradient descent algorithm allows one to have a greater control over the steps it takes whilst undergoing the optimisation process. In the custom optimisation algorithm proposed, a batch of random samples is picked per epoch, their distances computed, and last the gradient of the matrix $\mathbf{A}$ computed. With the gradient, $\mathbf{A}$ itself gets updated in the direction of gradient descent. Every 10 epochs, the total loss is computed with the validation set; if it is smaller than the previous, it is allowed to continue running, otherwise the learning rate is halved. Using the validation cost as a more accurate performance measure of the algorithm allows a good way of determining the optimum $\mathbf{A}$ to use for separation of the data points.

The learning rate, batch size and number of epochs were chosen by performing a coarse manual search, and it was determined best to use 10^{-3}, 50 and 100 respectively. The starting matrix $\mathbf{A_{init}}$ was chosen to be the identity matrix, due to the lack of a better estimate. Note that this matrix is important, as due to the nature of the loss function, falling into local minima depends greatly on the starting point of the algorithm.

The first implementation, minimising Equation 6, computes the cost from a random subset of points picked from the entire validation set. The points in each batch are also randomly picked from the training set. The loss on the validation set decreases linearly, and is unbounded, Figure 6. This is not desired, as no optimal $\mathbf{A}$ can be found. This nevertheless indicates, that the optimisation algorithm is only focusing in pulling apart dissimilar points, as the loss function does not penalise such behaviour. Even though the validation loss is not of much value, the $\mathbf{A}$ computed is still useful, since the points are pulled apart in such a way that improves the rank-k performance. Figure 2b shows the rank-k accuracy after 200 epochs.

The second implementation introduces further constraints to make sure similar points are being clustered, minimising Equation 7. The triplets point set are recomputed every time the cost is calculated, to make sure, the algorithm is always optimising points that need it. The validation set also uses a set of triplets, but in this case it is fixed throughout the optimisation process. This proposal yields again higher accuracies than the baseline in terms of mAP, see Figure 2b and Table 2d, but these are still minimal.

Contrary to the previous implementation the validation cost for this model increases after every epoch stabilising after $\sim$ 70 epochs (Figure 6, signifying the stabilisation of the model. This very likely signifies the overfitting of the model to the training data which is already very well clustered. The ever increasing validation cost of the second implementation suggests the model is overfitting to the training data and hence is performing worse when subject to data it hasn't seen before. This would explain the linear ever decreasing behaviour of the validation cost of the first implementation, which signifies the spreading of already separate dissimilar points even more whilst leaving the actual impostors and distant similar points untouched.

The overfitting of the model would imply lower epoch values of $\mathbf{A}$ would perform better than higher value ones (Figure 6 - Appendix II). Comparing different epoch values in Figure 9 (Appendix II) it can be seen how in fact lower epoch valued matrices give slightly better results on the test data than the overfitting higher valued ones.

Starting the optimisation from the identity matrix causes the found transformation matrix to be largely similar to the identity, due to the algorithm hitting a local minima. Even though this might prevent the algorithm finding a better minimum, it is better than finding a local minima that corresponds to a matrix that overfits the data.

6. Conclusion

This report has aimed at describing different models in the problem of person re-identification, and has discussed the performance of the implementations. The dataset feature vectors were obtained by training the ResNet50 with triple loss on CUHK03 images, and so are already very well clustered by classes. The proposed improved methods achieve marginal improvements and suffer from data overfitting. This suggests the dataset itself is constraining the models to separate data, with the training data being very well clustered whilst the test data providing more of a challenge, hence making the model prone to overfitting due to the *optimality* of the training data. This report concludes that even though the improved methods do marginally improve the metric learning objective, they are very susceptible to overfitting, and excessive training on the training data does in fact decrease its accuracy on test data.

References

[1] Jonathan Hui. *mAP (mean Average Precision) for Object Detection*. 2018. URL: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173.

[2] Wikipedia. *Mahalanobis distance*. 2018. URL: https://en.wikipedia.org/wiki/Mahalanobis_distance.

[3] K.Q. Weinberger and L.K. Saul. "Distance metric learning for large margin nearest neighbor classification". In: *Journal of Machine Learning Research* 10 (2009), pp. 207–244. ISSN: 15324435.

[4] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. 2017. arXiv: 1703.07737 [cs.CV].

[5] CJ Carey and Yuan Tang. *metric-learn*. 2015. URL: github.com/metric-learn/metric-learn.

7. Appendix I: Gradient Derivation

The first step is to redefine the problem so we can do the product rule on the distance function. Below is shown the mapping and naming used in the derivation.

$$d_{\mathbf{A}}(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} - \mathbf{x_j})^T \mathbf{A} (\mathbf{x_i} - \mathbf{x_j})$$
$$Let\ \mathbf{y} = (\mathbf{x_i} - \mathbf{x_j}) \quad \& \quad \mathbf{z} = \mathbf{A}\mathbf{y}$$

We will now use the product rule as shown below to compute the derivative in two steps.

$$\frac{\delta(d_{\mathbf{A}})}{\delta \mathbf{A_{ij}}} = \frac{\delta \mathbf{y^T z}}{\delta \mathbf{A_{ij}}} = \frac{\delta \mathbf{y^T z}}{\delta \mathbf{z_k}} \frac{\delta \mathbf{z_k}}{\delta \mathbf{A_{ij}}}$$
$$= \sum_k \frac{\delta \mathbf{y^T z}}{\delta \mathbf{z_k}} \frac{\delta \mathbf{z_k}}{\delta \mathbf{A_{ij}}}$$

We will first compute the derivation of $\mathbf{y^T z}$ with respect to $\mathbf{z_k}$, where $\mathbf{z_k}$ is the k^{th} element of $\mathbf{z}$.

$$\frac{\delta \mathbf{y^T z}}{\mathbf{z_k}} = \frac{\delta \sum_p \mathbf{y_p z_p}}{\delta \mathbf{z_k}}$$
$$= \sum_p \frac{\delta \mathbf{y_p z_p}}{\delta \mathbf{z_k}} = \mathbf{y_p} \delta_{pk}$$
$$= \mathbf{y_k}$$

The above derivation shows how $\mathbf{y^T z}$ is the inner product of both vectors and so is the sum over the element-wise multiplication. The next step simply states how the derivative of the multiplication will only depend on the element $\mathbf{z_k}$ if $p = k$, hence the δ_{pk}. As a result the derivative of $\mathbf{y^T z}$ with respect to the elements of $\mathbf{z}$, is the k^{th} element of $\mathbf{y}$, $\mathbf{y_k}$.

$$\frac{\delta \mathbf{z_k}}{\delta \mathbf{A_{ij}}} = \sum_m \frac{\delta \mathbf{A_{km} y_m}}{\delta \mathbf{A_{ij}}}$$
$$= \delta_{ki} \delta_{mj} \mathbf{y_m} = \delta_{ki} \mathbf{y_j}$$

To obtain the derivative of the elements of $\mathbf{z}$ with respect to the elements of $\mathbf{A}$, we first have to realise $\mathbf{z_k} = \sum_m \mathbf{A_{km} y_m}$, that is the sum of the multiplication between the elements of $\mathbf{y}$ and of the elements of row k of matrix $\mathbf{A}$. From that we again take the sum out of the derivative and realise the value of $\mathbf{z_k}$ will only vary if the matrix values match, namely $k = i$ & $m = j$. From that we get to the final solution $\delta_{ki} \mathbf{y_j}$.

$$\frac{\delta(d_{\mathbf{A}})}{\delta \mathbf{A_{ij}}} = \frac{\delta \mathbf{y^T z}}{\delta \mathbf{z_k}} \frac{\delta \mathbf{z_k}}{\delta \mathbf{A_{ij}}} = \delta_{ki} \mathbf{y_k y_j} = \mathbf{y_i y_j}$$
$$\frac{\delta(d_{\mathbf{A}})}{\delta \mathbf{A}} = \mathbf{y y^T}$$

Finally joining both solutions we arrive a t the final solution, which shows the derivative of the distance with respect to matrix $\mathbf{A}$ is the outer product of the difference vector $\mathbf{y}$.

8. Appendix II

8.1. Baseline model visual representation

Figure 3: Example of Rank-10 for 5 different query images. The first image to the left is the query image, and the ten that follow are the first ten neighbours(red implies different label, green same label).

8.2. K-means Clustering Ranklist Accuracies

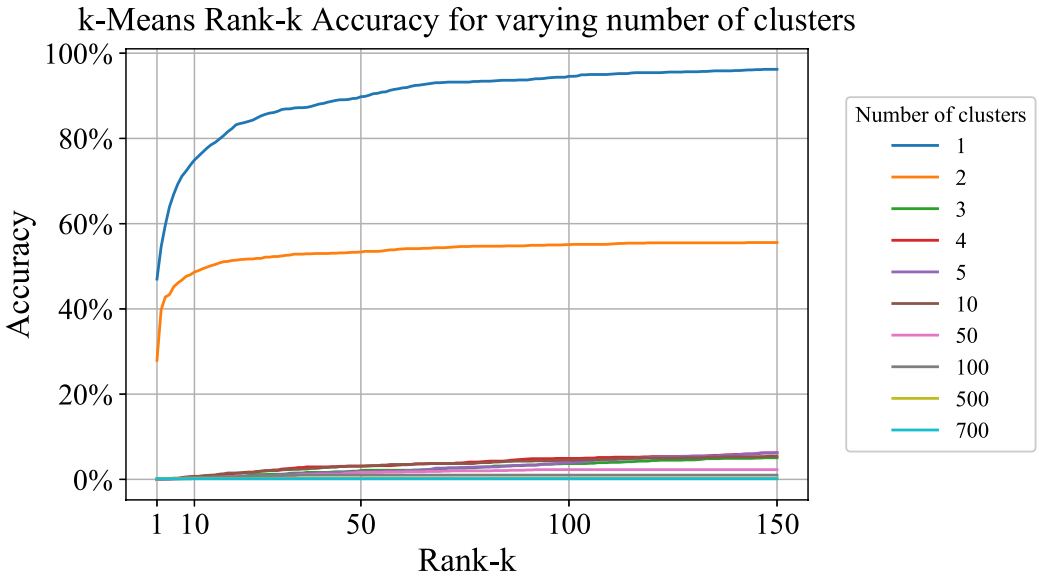


Figure 4: Rank-k accuracy for k-Means clustering baseline method.

8.3. Mahalanobis Accuracies

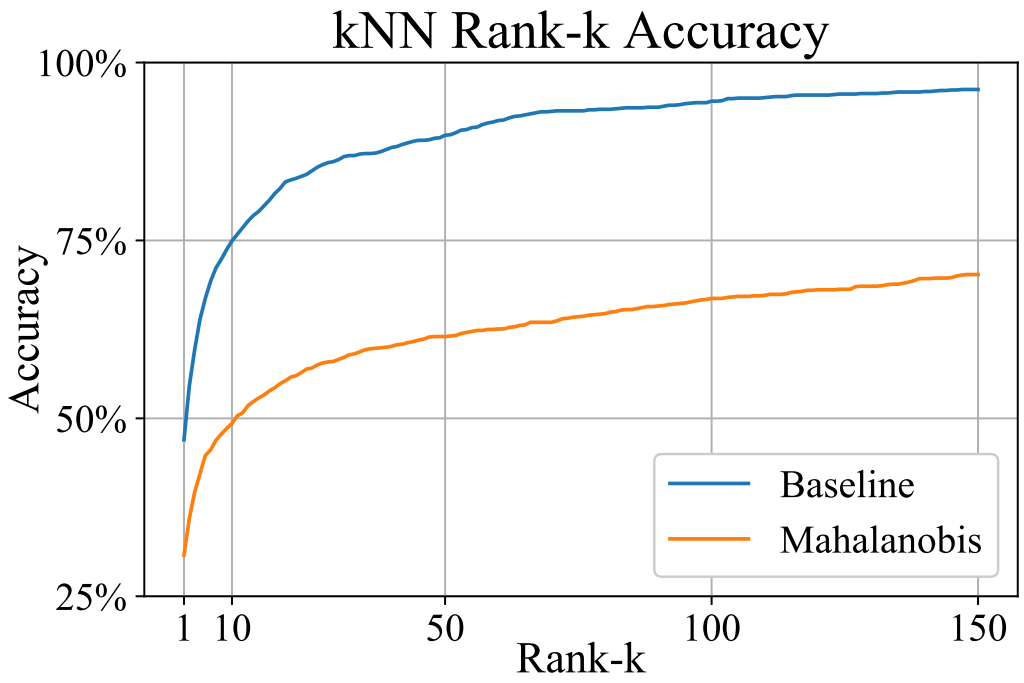


Figure 5: Rank-k accuracy for Mahalanobis distance

8.4. Validation Cost for Proposed methods

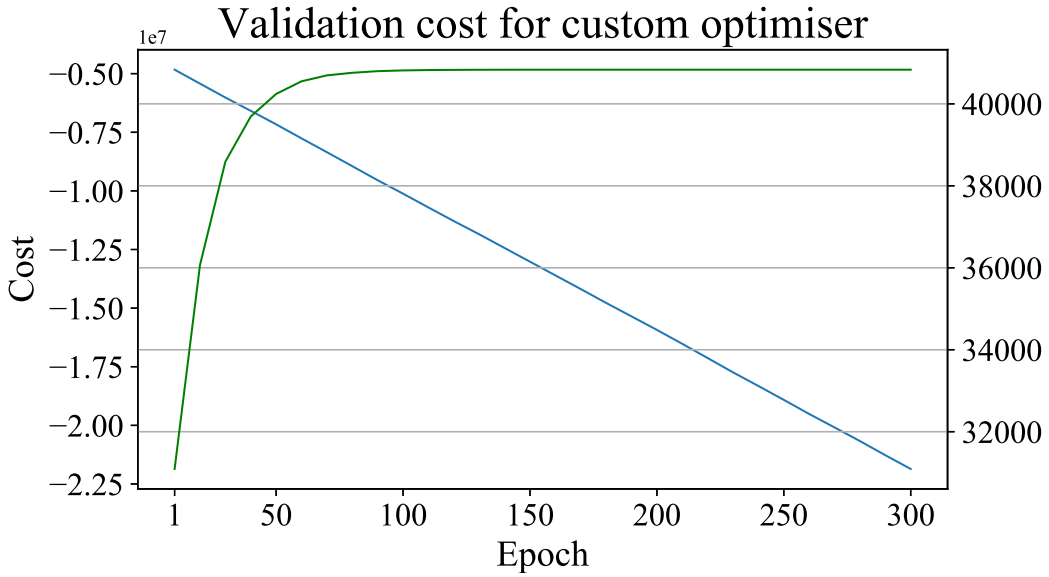


Figure 6: Validation Cost against epochs for proposed methods 1 (blue) and 2 (green)

8.5. Proposed method performance for different epochs

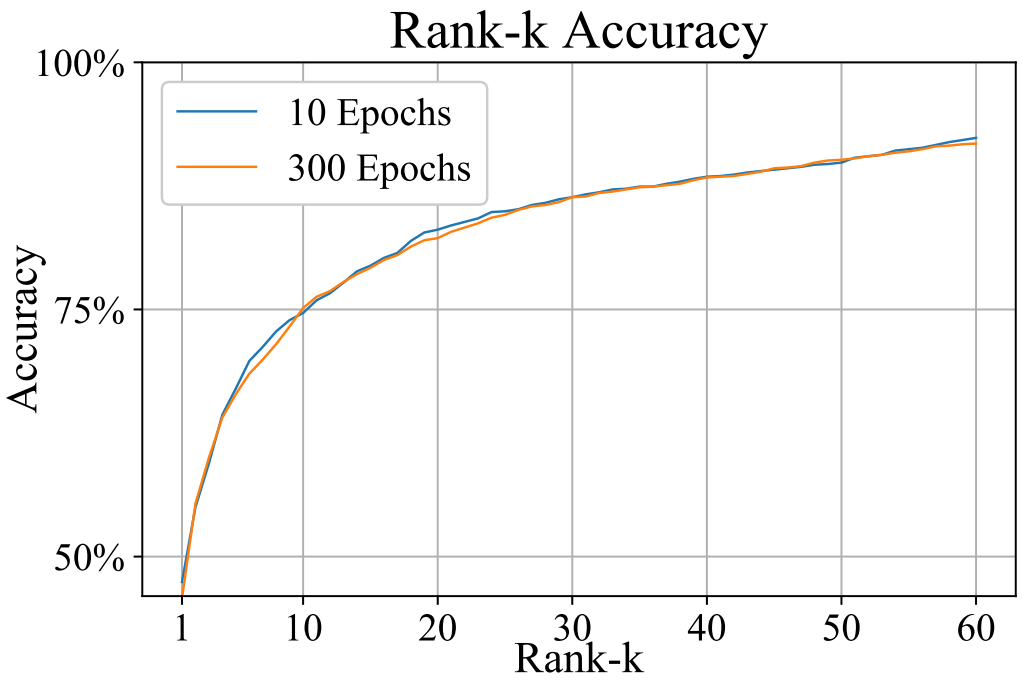


Figure 7: Rank N performance of the optimiser after 10 (blue) and 150 (red) epochs. The model with after more epochs performs slightly worse than the one after less epochs.

9. Appendix III: Results

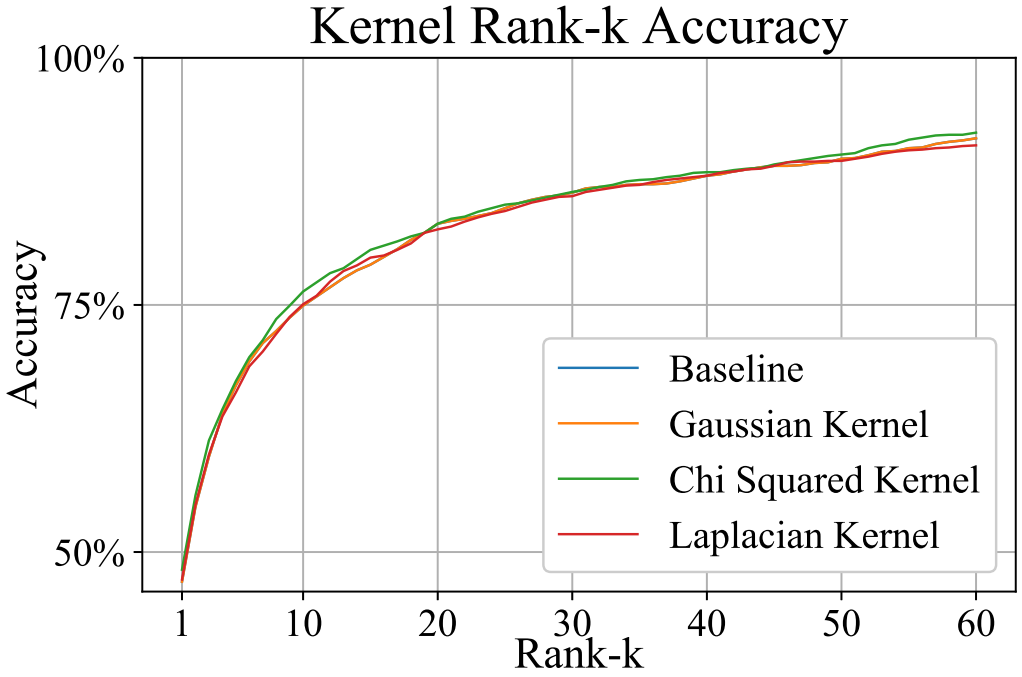


Figure 8: Kernel method

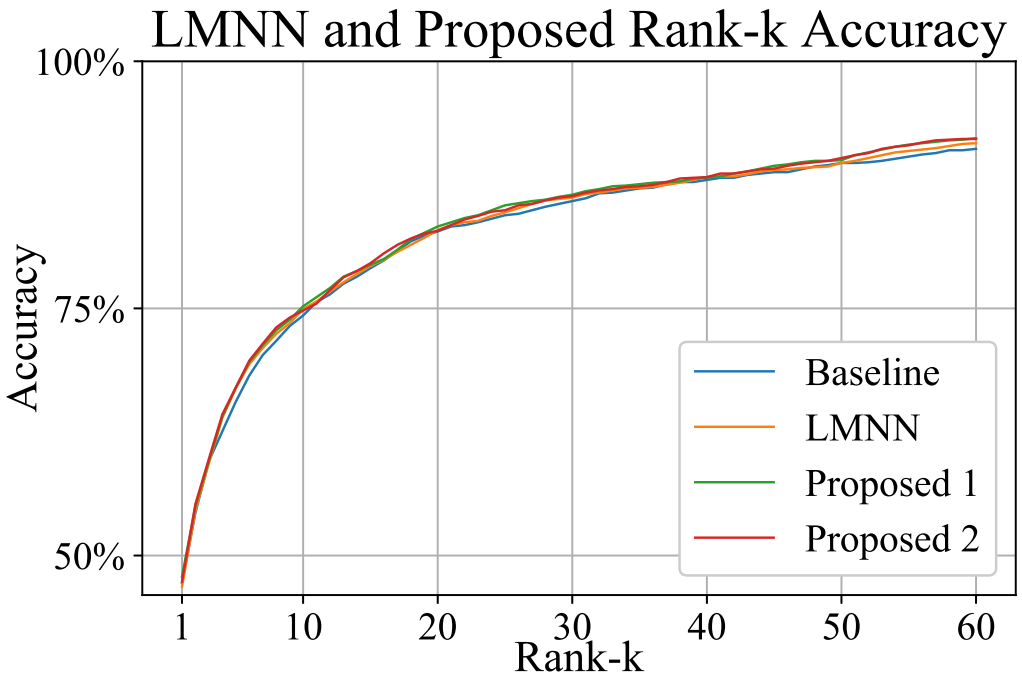


Figure 9: LMNN and custom methods

10. Appendix IV: Code instructions

Code: Link
The dataset has to be located in a folder called 'PR_data' in the same folder as the python files.
The code of each section of this report is divided into python files as follows:

- There are three python files that define required classes for code operation. These are:
  - DataClass.py: Loads dataset data from source and splits into required sets.
  - NearestNeighbors.py: Implements kNN and computes accuracies (plus other functionality, such as kernel implementation and feature transforms). This class depends on the sklearn library.
  - Optimiser.py: Implements the custom optimiser discussed in this report (both variants).
- Note that many of the following files will save data to pickle files, as to be able to restore that information after the code execution or in-between code calls. The information is stored into the folder 'Generated Data' that has to exist on the same path as the python files. (This file should already exist in the zip file provided)
- kNN baseline: baselineMain.py
  - This file computes the rank-k accuracies for kNN, using the Euclidean, Manhattan and Cosine distances.
  - It plots the accuracies, and generates a second plot with the correct/wrong matches with images for reference.
- kMeans baseline: kMeansMain.py
  - This file computes the kMeans baseline.
  - Note, this file (and all future ones) loads the dataset from a pickle file that is generated in 'baselineMain.py', for faster operation of the code. This default operation can be reversed by modifying the python file as instructed in its comments.
- Kernel approach: kernelMain.py
  - This file contains the kernel approach.
- Mahalanobis: mahalanobisMain.py
  - This file contains the Squared Original Mahalanobis implementation.
- LMNN from library: LMNNMain.py

  - This file contains the implementation of LMNN from the library metric-learn. This library must be installed for the code to execute.
  - The model will be fitted for different values of the regulariser and k.
  - The model will not be plotted, this is done later along the optimser plots.
  - Note that this file takes long to execute.

- Custom optimiser: customOptimiserMain.py
  - This file calls the optimisation class.
  - If when calling the optimisation class 'useTripplets' is set to false, the first variant discussed in this report is executed, otherwise the second variant.
  - When running the optimiser, every 10 epochs a pickle file is saved with the current state of the transformation matrix A. For plotting, the path to these matrices has to be specified in the locations indicated in the code (path1 and path2 respectively).

- MiscLibrary.py and ValidationPlot.py are used to generate plots.