Master's Degree

Computational Engineering and Intelligent Systems

(Department of Computer Science and Artificial Intelligence)

Master thesis

Face Beauty Analysis via Manifold based Semi-Supervised Learning

Anne Elorza Deias

Directors

Fadi Dornaika

Ignacio Arganda Carreras

Acknowledgements

I would like to thank my directors, Fadi and Ignacio, for their sensible and practical advices, as well as their extraordinarily prompt responses to all my e-mails, which must not be taken for granted. My family and closest friends, who are not madly keen on science, and nonetheless listened attentively to the finest details of this work (or so did they say). I especially thank Maria, for understanding, Tonina, for trying to understand, Izaskun, for being there, Uxue, for popping up from time to time, Lierni and Gaizka, for teaching me the topology of knots and ropes, and Asier, for our nocturnal discussions about Lebesgue's integral in neural networks.

Summary

Beauty has always played an important role in society, implicitly influencing the human interactions of our daily lives and more significant aspects, such as the mate choice or job interviews. And now, with the progress made in deep learning and feature extraction, automatic facial beauty analysis has become an emerging research topic too. However, the subjectivity of beauty still hinders the development in this area, due to the cost of collecting reliable labeled data, since the beauty score of an individual has to be determined according to various raters.

To address this problem, we study the performances of four different semi-supervised manifold based algorithms, which can take advantage of both labeled and unlabeled data in the training phase, and we use them in two different datasets: SCUT-FBP and M^{2}B. The learning algorithms are Local and Global Consistency, Flexible Manifold Embedding and Kernel Flexible Manifold Embedding. There is an additional algorithm, which, unlike the rest of them, instead of performing classification, obtains a non-linear transformation of the data to make the classification easier. All of these algorithms were designed to work on discrete classes, but we perform regression, where labels are real numbers. So the first step, in chapter 2, is to analyse how the algorithms can be adapted to regression and to hypothesize which problems we could be encountering in this process. Secondly, we empirically test them (chapter 3). The best results are obtained with KFME on both datasets, achieving a mean average error of 0.0550 (out of 1) and a Pearson correlation of 0.8446 on SCUT-FBP dataset. With respect to M^{2}B dataset, a mean average error of 0.1357 and a Pearson correlation of 0.4469 are achieved on eastern faces, while a mean average error of 0.1132 and a Pearson correlation of 0.6322 are achieved on western faces. This dissertation ends with a final chapter discussing the results and proposing new topics of study for future work.

Contents

Chapter 1

Introduction

1.1 Motivation

This work is a comparative study of manifold based semi-supervised algorithms applied to the specific problem of automatic facial beauty assessment. The statement of the problem can be the following: given a facial image, what would its beauty score, ranging from 1 to 10, be? Its resolution, instead, is a highly complex matter.

The main problem we will try to address is the scarcity of labeled data regarding this specific topic, which is due to its subjective nature. It is well known that beauty lies in the eye of the beholder, so obtaining reliable databases requires a great deal of human labor. In order to obtain meaningful labels, many raters are needed and, usually, the ground truth label of an image is considered to be the average score of all the ratings given by different raters. So, in this context, semi-supervised learning appears to be particularly appropriate. Unlike supervised learning, which just makes use of labeled data, this approach utilizes the information underlying the unlabeled data as well. There exist several ways of taking advantage of unlabeled samples. In our case, we have chosen graph-based methods, which, roughly speaking, work under the assumption that similar samples should share similar labels. Thinking about beauty, it seems quite reasonable to assume that when two faces resemble each other they should be similarly attractive. This abstract idea can be materialized by creating a fully-connected weighted graph, in which nodes are samples and the weights between each pair of nodes represent their similarities. Chapter 2 specifies this mechanism in greater detail.

As one can imagine, automatic beauty evaluation is not limited to exploiting as much information as one can and many other issues arise during its study. For example, a crucial aspect is which features represent best a face or, if one prefers philosophy over technicity, they may wonder whether a machine will ever be able to learn something so seemingly human as beauty perception. In the next section, a brief review will be presented, and we will see that these queries and others have already been made in the past.

1.2 Related work: Beauty analysis in computer science

The first automatic facial beauty system [1], published in 2001, aimed at classifying beauty in 4 different classes. Each image was described by 8 ratios (e.g. the ratio of the horizontal distance between the eyes to the average vertical distance between the eyes and the mouth), which were supposed to capture the essence of beauty according to previous studies made by psychologists and biologists. This means there were only 8 features, and the performance of the classifier strongly depended on a correct localization of the eyes and the mouth. The dataset consisted of 80 photos, each of them had been rated by 12 individuals among the 4 classes, and the ground truth label was the median of all raters' ratings. The classifyer was a variant of k-nearest neighbors.

Since then, many aspects of the research have been developed. The first accurate facial attractiveness predictor [9, 10] used a high number of features: 98 image features, 90 geometric and 8 related to face simmetry, hair and skin color and skin smoothness. Their study, along with others [4], reinforced the idea that beauty perception is something a machine can learn. However, together with the still limited size of the dataset (around 90 images), this study still had another major flaw: the process of feature extraction was not fully automatic. In fact, the geometric features were based on landmarks. These were found by an automatic engine capable of identifying eyes, nose, lips, eyebrows, and head contour; yet some of them failed to be correctly identified and had to be adjusted manually. This drawback was firstly overcome by building a family of convolutional neural networks whose input was the raw image and the output was the score [7].

One of the principal challenges of the research in this field has been to build accurate facial representations, which can be either feature-based, holistic or hybrid. In the first category one can find geometric, color, texture or other local representations. Geometric representations use landmark points and consider their positions, distances between them or ratios of these distances, and some of the studies have also focused on the relationship between the golden ratio and beauty [16]. On the other hand, the holistic approach uses the global information of the face, instead of local features, e.g., using eigenfaces [4]. This approach also includes recent representations learnt with deep learning [6, 21].

Since 2013, the studies in this area have partly focused on constructing larger and reliable benchmarks [13, 12, 22]. Many recent works make use of various neural networks [6, 13, 22, 23]. One paper utilizes semi-supervised learning [6]; more specifically, it uses deep self-taught learning. This strategy combines the semi-supervised setting with deep learning in order to extract more meaningful features, whereas the classification remains supervised.

If one wishes a thorough insight into the related work, a complete review can be found at [11].

1.3 Our study

The problem we will be facing will be regression, where labels belong to a continuous space, namely the interval of possible scores, rather than a restricted set of discrete classes. For this purpose we will use four semi-supervised manifold based algorithms (see chapter 2) and a couple of classical supervised algorithms to establish the comparison between them, all of which have been implemented in MATLAB[1]. The two datasets used in the study are described below.

1.3.1 Datasets

Two different datasets are used in this work: SCUT-FBP dataset [22] and the Multi Modality-Beauty (M^2B) dataset [13]. The former was specifically designed for automatic facial beauty perception and contains high resolution front-on face portraits of Asian females. On the other hand, the latter was developed to evaluate beauty via face, dressing and/or voice on both Eastern and Western females and each instance in the dataset contains information about the three modalities. However, we are only focusing on the facial images, which unlike the ones in SCUT-FBP dataset, show very different poses and expressions. This complicates, in consequence, the beauty assessment, which could be found difficult even by a human rater.

SCUT-FBP dataset

SCUT-FBP dataset contains 500 high resolution front-on face portraits of Asian females with neutral expressions, simple background and minimal occlusion, as can be seen in figure 1.1. These characteristics prevent from taking into account irrelevant factors in the beauty classification task.

The beauty rankings lie on the interval (1, 5) and are the result of averaging various ratings. The ratings were collected among 75 individuals using a web based tool, as can be seen in figure 1.2, with an average number of 70 raters per image.

The ratings aproximately follow a normal distribution (figure 1.3) with a small peak around 4.5. This suddenly high number of beautiful individuals is "artificial". Even though in the real world beautiful faces constitute a small percentage of the population, more beautiful faces were included in the database to facilitate the beauty classification task.

Raters' consistency and self-consistency are checked in different ways by the authors of the paper. For instance, low standard deviations in the ratings of each image indicate rater's agreement in the perception of beauty.

[1]© 2017 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `https://es.mathworks.com/company/aboutus/policies_statements/trademarks.html` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

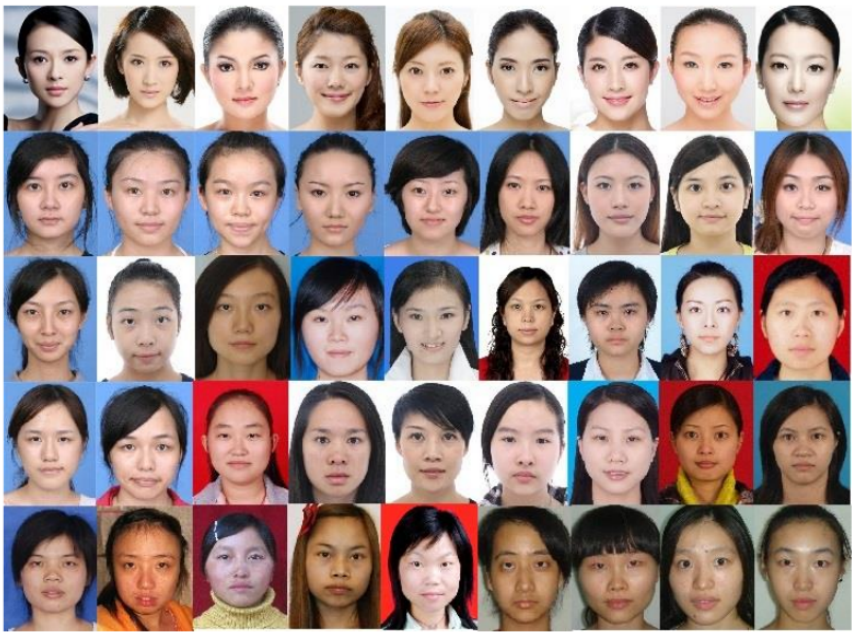
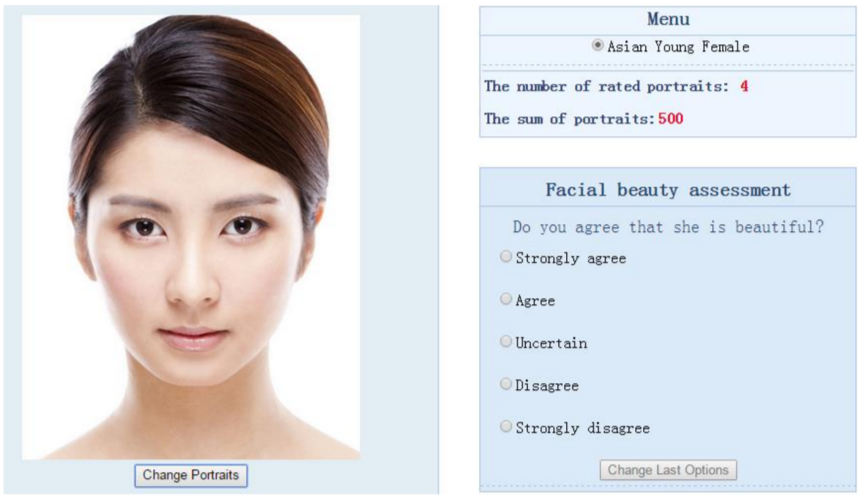


Figure 1.1: Examples of face portraits of SCUT-FBP dataset from [22].

M^{2}B dataset

The Multi-Modality Beauty dataset has been developed to study beauty perception in three different modalities, in dressing, in the face and in the voice, as well as the global beauty perceived when any of these three aspects are combined. Therefore, the dataset contains one face photo, one full body photo and one voice snippet of 1240 females belonging to two ethnic groups: westerners and easterners (620 individuals in each group). In addition, each of the females of the dataset is rated, in the different modalities and their combinations, with various scores in the interval [1, 10].

The ratings were collected among 40 participants, which were split into two groups depending on their ethnicity, so that each of the participants rated females of their own ethnic group. The web tool used for this purpose can be seen in figure 1.4. The ratings were obtained using k-wise comparison, which means that the raters are asked to order k females according to their beauty, and then these k-wise ratings were converted into global ratings in the interval [1, 10] by solving an optimization problem to preserve as many pairwise preferences as possible. The drawback of this method of collecting the labels is that, opposedly to SCUT-FBP dataset, where we had the ratings of various raters per image, here we have a unique rating. Thus, we cannot really measure the uncertainty of each of the labels, even if it seems to be important, since beauty is not an absolute concept.



The number of rated portraits: 4

The sum of portraits: 500

Facial beauty assessment

Do you agree that she is beautiful?

- Strongly agree
- Agree
- Uncertain
- Disagree
- Strongly disagree

Figure 1.2: Facial beauty asses system from [22].

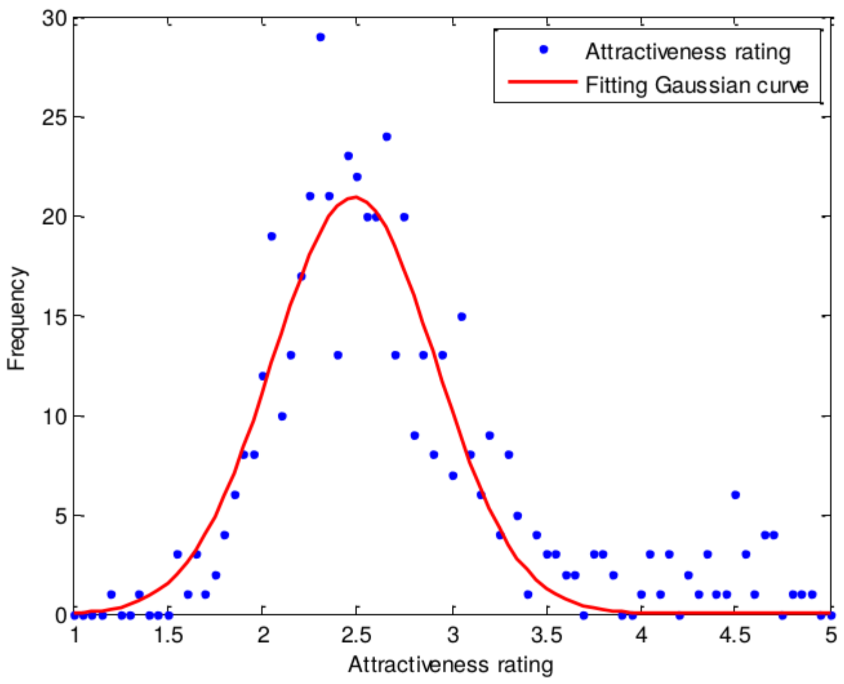


Figure 1.3: Histogram of the rating distribution from [22].

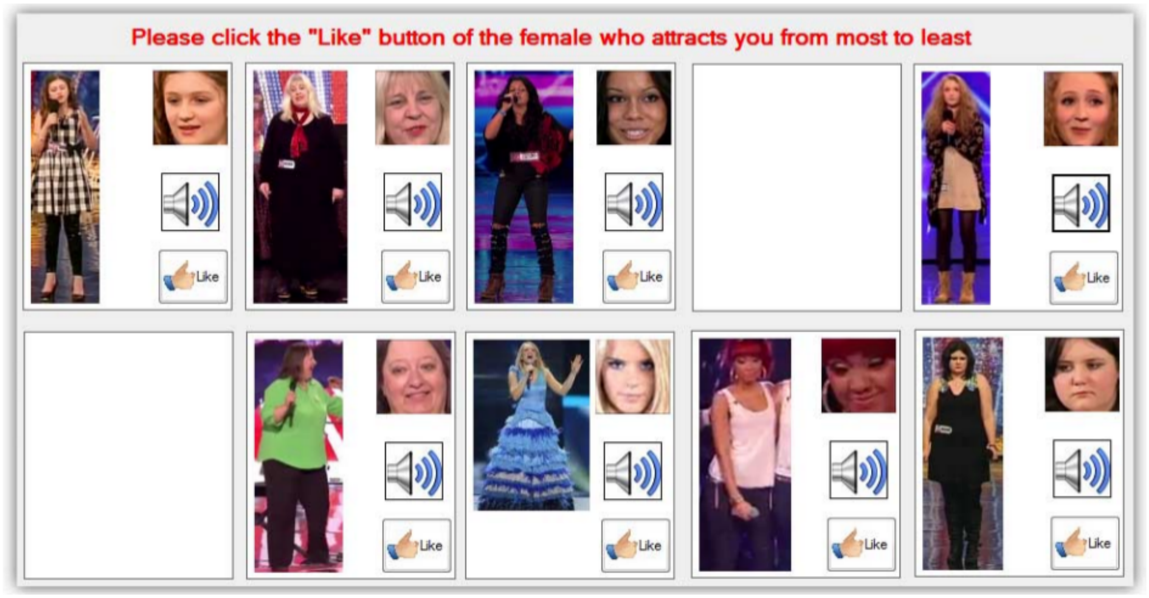

Please click the "Like" button of the female who attracts you from most to least
Like
Like
Like
Like
Like
Like
Like
Like

Figure 1.4: User interface of the attractiveness ranking tool from [13].

Chapter 2

Learning algorithms

In this chapter the mathematical background of the work is presented in two sections, the first of them regarding semi-supervised learning, where, appart from the learning algorithms, the construction of the similarity graph is explained. Secondly, the supervised algorithms are described. Most of the effort is put in the description of the optimization problem that each algorithm aims at solving, so that the reader can get the intuition of how the regression is approached and what role each of the parameters plays. On the contrary, little attention is paid to the intricacies of how exactly it is solved, so, for a deeper understanding of the specific functioning, one can see the references.

2.1 Semi-supervised algorithms

In this section, the semi-supervised algorithms used in this work are explained. As commented in the introduction, the key to exploiting the unlabeled data is the usage of similarity graphs. Even though we may not know the labels of the unlabeled data, we train the machine so that unlabeled instances get nearly the same labels as the labeled instances with which they share a high similarity. Three of the presented algorithms belong to the family of manifold based label propagation methods, which are named so because they aim at "propagating" labels from labeled data points to unlabeled data points according to the intrinsic manifold structures collectively revealed by all the data (we will explain, in the sequel, the meaning of this idea). The last algorithm, however, is only an embedding method which computes a non-linear transformation of the data and offers the possibility to reduce its dimensions just by removing a part of them, like in principal component analysis. Then, this process has to be combined with a learning algorithm to perform the regression. Before proceeding to the description of these algorithms, though, we will briefly explain the mathematical notation in which they are fundamented.

2.1.1 Mathematical notation of the semi-supervised algorithms

In what follows we will adopt these conventions. Matrices and vectors, as opposed to scalars, are noted with bold font, and they can be distinguished using capital or lowercase letters respectively.

$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_l, \mathbf{x}_{l+1}, \cdots, \mathbf{x}_{l+u}] \in \mathbb{R}^{D\times(l+u)}$ is the data matrix. D is the number of features representing each sample, l is the number of labeled instances, u the number of unlabeled instances and $N = l + u$ is the total number of samples. The data matrix is represented by columns, i.e., each $\mathbf{x_i}|_{i=1}^{N}$ is a column feature vector corresponding to a given sample. $\mathbf{x_i}|_{i=1}^{l}$ are the labeled samples, while $\mathbf{x_i}|_{i=l+1}^{l+u}$ are the unlabeled ones.

Since our problem consists in regression, the labels are real numbers and they can be represented as a column vector $\mathbf{y} \in \mathbb{R}^{N\times1}$. The first l rows of $\mathbf{y}$ will contain the labels of the labeled instances, while the last u rows will generally be 0, since they correspond to unlabeled data. $\mathbf{f} \in \mathbb{R}^{N\times1}$ will be the predicted labels. However, the semi-supervised algorithms we are using were originally developed for classification tasks, so, when describing classification, C will be the number of classes. The ground truth labels $\mathbf{Y}$ and the predicted labels $\mathbf{F}$ will be matrices in $\mathbb{R}^{N\times C}$, where $Y_{ij} = 1$ if sample i belongs to class j and $Y_{ij} = 0$ otherwise.

As already stated, the way of exploiting the information contained in unlabeled data is considering a similarity graph between samples. For this sake, we have to introduce a similarity matrix $\mathbf{S} \in \mathbb{R}^{N\times N}$ (which will be symmetric, so our graph has to be undirected). Each element S_{ij} of $\mathbf{S}$ is the similarity between samples i and j (the different similarities considered in this particular work will be described below). In addition, the Laplacian matrix of $\mathbf{S}$, $\mathbf{L}$, is used. $\mathbf{L} = \mathbf{D} - \mathbf{S}$, where $\mathbf{D}$ is the diagonal matrix whose elements are the row sums of $\mathbf{S}$. Besides, Local and Global Consistency uses the normalized Laplacian, $\hat{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{S}\mathbf{D}^{-1/2}$, where $\mathbf{I}$ is the identity matrix of size N.

Finally, $\mathbf{1}, \mathbf{0} \in \mathbb{R}^{N\times1}$ note vectors with all elements as 1 and 0 respectively. And everytime a norm $||\cdot||$ is used, it has to be understood as the usual norm, namely, the Euclidean norm.

2.1.2 Local and Global Consistency

Local and Global Consistency (LGC) [24] is the simplest algorithm we will be using, since it was one of the first graph-based label propagation methods. It aims at predicting the labels of all labeled and unlabeled instances, $\mathbf{F}$, by minimizing the following function:

$$q(\mathbf{F}) = \sum_{i,j=1}^{N} S_{ij} \left\| \frac{\mathbf{f}_i}{\sqrt{D_{ii}}} - \frac{\mathbf{f}_j}{\sqrt{D_{jj}}} \right\|^2 + \mu \sum_{i=1}^{N} \|\mathbf{f}_i - \mathbf{y}_i\|^2 , \qquad (2.1)$$

where D_{ii} is the sum of the i-th row of $\mathbf{S}$ or, in other words, the sum of the similarities of sample i with all the other images.

The solution to this problem can be found analytically, deriving correctly the cost function, and it is $(\mathbf{I} + \hat{\mathbf{L}}/\mu)^{-1}\mathbf{Y}$, where $\mathbf{I}$ is the identity matrix of N dimensions.

Once solved this problem and obtained a matrix $\mathbf{F}$, the predicted class of an instance i will be the maximum index j of the i-th row of $\mathbf{F}$.

Interpretation of the cost function

The first term is the smoothness constraint, the second term is the fitting constraint and μ is the parameter which controls the trade-off between them. Intuitively, the label smoothness tries to maintain the intrinsic structure of labeled and unlabeled samples (we will inmediately see how), while the label fitness is the square of the Euclidean distance between all the ground truth labels $\mathbf{Y}$ and all the predicted labels $\mathbf{F}$. So this second term just tries to adjust as well as possible the already known labels to the predicted ones. Simply put, this is the part in charge of the memorization of the known labels.

The label smoothness, obtained by the term

$$\sum_{i,j=1}^{N} S_{ij} \left\| \frac{\mathbf{f}_i}{\sqrt{D_{ii}}} - \frac{\mathbf{f}_j}{\sqrt{D_{jj}}} \right\|^2 ,$$

is what allows the algorithm to learn from the unlabeled data as well. If this term did not exist, the algorithm would just memorize the already existing labels. Generally speaking, the label smoothness is what makes similar instances share similar labels, although this is not precisely what the algorithm does. It is exactly what would happen if D_{ii} and D_{jj} did not appear in the expression. Why? Assuming that D_{ii} and D_{jj} did not exist, the label smoothness would be

$$\sum_{i,j=1}^{N} S_{ij} \left\| \mathbf{f}_i - \mathbf{f}_j \right\|^2 .$$

If two images were very similar, that is if S_{ij} was nearly 1, the difference between their predicted labels $\mathbf{f}_i$ and $\mathbf{f}_j$ would be more penalized because it is pondered by S_{ij}, whereas if they were very dissimilar, which happens when S_{ij} tends to be 0, them having very different labels would almost not increase the cost.

So what is the effect of adding the term D_{ii} to the equation? Let i_1 and i_2 be two labeled instances, the first of them with a high $D_{i_1 i_1}$ associated while the second one has a low $D_{i_2 i_2}$ associated. Supposing there is an unlabeled instance j very similar to both of them, that is $S_{i_1 j}$ and $S_{i_2 j}$ are nearly 1, what happens to the predicted label of sample j? Since $D_{i_1 i_1}$ is high, $\mathbf{f}_{i_1}/\sqrt{D_{i_1 i_1}}$ will be low, and, hence, when minimizing

$$\sum_{i,j=1}^{N} S_{ij} \left\| \frac{\mathbf{f}_i}{\sqrt{D_{ii}}} - \frac{\mathbf{f}_j}{\sqrt{D_{jj}}} \right\|^2 ,$$

the influence of i_1 over j will be moderate, since $\mathbf{f}_j/\sqrt{D_{jj}}$ will tend to be proportional to $\mathbf{f}_{i_1}$ but having a small value. On the contrary, since $D_{i_2 i_2}$ is low, $\mathbf{f}_{i_2}/\sqrt{D_{i_2 i_2}}$ will be high and thus this term will be very influential with respect to $\mathbf{f}_j$.

Now, recalling the definition of D_{ii} (it is the sum of the i-th row in $\mathbf{S}$), we deduce that the most influential samples are the ones being the least similar to the rest.

A simple example to understand the role of $\mathbf{D_{ii}}$

To illustrate this explanation, we may consider a very simple similarity graph (see figure 2.1).

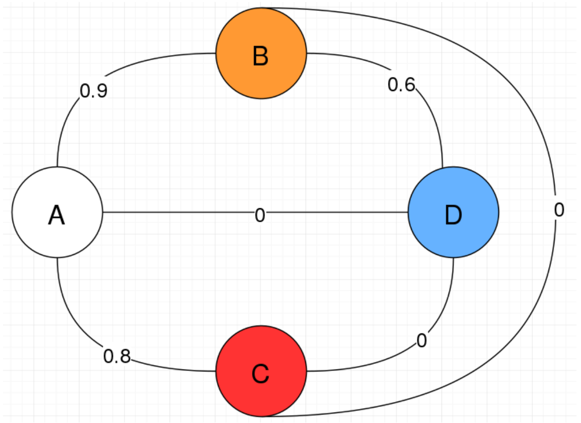


Figure 2.1: A simple example to understand the role of D_{ii} in the cost function.

In this example, we can find four individuals A, B, C and D and the similarities between them. Each color represents a class, except white color, which indicates that a node is unlabeled. In our case, A is the only unlabeled node and the rest of them belong to different classes. The weights of the edges represent the similarities. One could expect node A to be classified as belonging to the same class as B, since it is the one with which shares the highest similarity. However, whenever the parameter μ in (2.1) is high enough to predict correctly all the labeled data, node A is always classified in node C's class. This is because C is more influential than B, since the sum of the similarities of C with the other nodes is only 0.8, while B's is 1.5.

Local and Global Consistency and regression

So far, we have commented on the functioning of the algorithm when it acts as a classifier, but our problem will be regression. So the natural question is whether it will still work as a regressor and how it should be modified for that purpose. Let $\mathbf{y}$ be the vector of ground truth labels, where unlabeled instances are labeled as 0, and $\mathbf{f}$ the predicted labels. Then equation (2.1) would become

$$q(\mathbf{f}) = \sum_{i,j=1}^{N} S_{ij} \left(\frac{f_i}{\sqrt{D_{ii}}} - \frac{f_j}{\sqrt{D_{jj}}} \right)^2 + \mu \sum_{i=1}^{N} (f_i - y_i)^2. \tag{2.2}$$

Here, we face two problems, which did not exist with classification. The simplest one is due to the label fitness term, $\sum_{i=1}^{N}(f_i - y_i)^2$. If an unlabeled instance i is labeled as $y_i = 0$, then f_i will tend to be 0, hence all the predicted labels will tend to be really small numbers. In the case of classification this did not happen or, being more precise, this happened, but it was irrelevant. Why? Because $\mathbf{F}$ was a matrix and the label of a sample i was the index of the highest element in $\mathbf{f}_i$, so the absolute values of $\mathbf{f}_i$ were irrelevant. However, in regression, the predicted label of an instance i is exactly f_i, so having smaller values of f_i when i is unlabeled than when it is labeled becomes a problem. A way of solving this, which we will adopt, is by setting y_i to the mean of the labeled instances when i is unlabeled.

The second problem regards the label smoothness,

$$\sum_{i,j=1}^{N} S_{ij} \left(\frac{f_i}{\sqrt{D_{ii}}} - \frac{f_j}{\sqrt{D_{jj}}} \right)^2,$$

and lies on the terms D_{ii} and D_{jj}. Suppose, as before, that i and j are a labeled and an unlabeled instance respectively and that $S_{ij} \simeq 1$. If D_{ii} is a large number, then $f_i/\sqrt{D_{ii}}$ will be low and, therefore, $f_j/\sqrt{D_{jj}}$ and, consequently, f_j will be low. A way of tackling this problem consists in normalizing the rows of the similarity matrix $\mathbf{S}$, i.e., dividing each row by its sum so that $D_{ii} = 1$ for each i. However, this means that the resulting matrix $\mathbf{S}_{norm}$ could not be symmetric and, thus, the algorithm cannot be applied. To satisfy the requirement of symmetry, we compute the following operation:

$$\frac{\mathbf{S}_{norm} + \mathbf{S}_{norm}^T}{2},$$

and use this similarity matrix instead of $\mathbf{S}$, whose row sums are roughly 1.

Although these two drawbacks could make the algorithm not suit perfectly our problem, it is interesting having this algorithm as a starting point to compare its performance with the more sophisticated versions of label propagation explained below.

2.1.3 Flexible Manifold Embedding

Similarly to LGC, Flexible Manifold Embedding (FME) [14] estimates the labels $\mathbf{F}$ by minimizing the following cost function:

$$g(\mathbf{F}, \mathbf{W}, \mathbf{b}) = \operatorname{tr}(\mathbf{F}^T\mathbf{L}\mathbf{F}) + \beta \operatorname{tr}[(\mathbf{F}-\mathbf{Y})^T\mathbf{U}(\mathbf{F}-\mathbf{Y})] + \mu(||\mathbf{W}||^2 + \gamma||\mathbf{X}^T\mathbf{W} + \mathbf{1}\mathbf{b}^T - \mathbf{F}||^2), \tag{2.3}$$

where $\mathbf{U}$ is an indicator matrix, that is a diagonal matrix, with its first l diagonal elements, corresponding to labeled instances, equal to 1, while the last u diagonal elements, corresponding to unlabeled instances, are equal to 0. W and b determine the unknown linear regressor which maps the original samples to the label space.

As with LGC, a closed-form solution, which is the one we will be using, can be found by setting the derivatives of g with respect to $\mathbf{W}$, $\mathbf{b}$ and $\mathbf{F}$ as 0 (for more details on how to obtain it, see [14]). The solution would be

$$\mathbf{b} = \frac{1}{l+u}\left(\mathbf{F}^T\mathbf{1} - \mathbf{W}^T\mathbf{X}\mathbf{1}\right)$$

$$\mathbf{W} = \gamma(\gamma\mathbf{X}\mathbf{H}_c\mathbf{X}^T + \mathbf{I})^{-1}\mathbf{X}\mathbf{H}_c\mathbf{F}$$

$$\mathbf{F} = \beta(\beta\mathbf{U} + \mathbf{L} + \mu\gamma\mathbf{H}_c - \mu\gamma^2\mathbf{Q})^{-1}\mathbf{U}\mathbf{Y},$$

with $Q = \mathbf{X}_c^T\mathbf{X}_c(\gamma\mathbf{X}_c^T\mathbf{X}_c + \mathbf{I})^{-1}$ and $\mathbf{H}_c = \mathbf{I} - (1/(l+u))\mathbf{1}\mathbf{1}^T$.

Flexible Manifold Embedding and regression

As with LGC, this algorithm was originally designed for classification tasks. Nonetheless, it can easily be adapted to work as a regressor. If $\mathbf{f}$ and $\mathbf{y}$ are real-valued vectors representing the predicted labels and the ground-truth labels, the cost function becomes

$$g(\mathbf{f}, \mathbf{w}, b) = \mathbf{f}^T\mathbf{L}\mathbf{f} + \beta(\mathbf{f} - \mathbf{y})^T\mathbf{U}(\mathbf{f} - \mathbf{y}) + \mu(||\mathbf{w}||^2 + \gamma||\mathbf{X}^T\mathbf{w} + \mathbf{1}b - \mathbf{f}||^2). \quad (2.4)$$

Interpretation of the cost function

The first term controls the label smoothness, the second one the label fitness and the last term fits a linear regression between features and labels, where $||\mathbf{w}||^2$ is a regularaztion term controlling the complexity of the model (thus, avoiding overfitting). β, μ and γ are the parameters controlling the trade-off between all the terms.

In spite of having been analysed in the case of LGC, the first two terms deserve further comment, since they present certain variations with respect to the previous algorithm. If we develope the expression $\mathbf{f}^T\mathbf{L}\mathbf{f}$, it would be

$$\sum_{i,j=1}^{n} S_{ij}(f_i - f_j)^2.$$

This is the same as in LGC if we remove the terms D_{ii} and D_{jj}. So, as explained before, this term is responsible of making similar images have similar labels.

The second term, the label fitness, $(\mathbf{f} - \mathbf{y})^T\mathbf{U}(\mathbf{f} - \mathbf{y})$ would be

$$\sum_{i=1}^{l}(f_i - y_i)^2,$$

whereas in the case of LGC it was

$$\sum_{i=1}^{N}(f_i - y_i)^2.$$

So the two problems that arised in the adaptation of LGC to regression vanish in the case of FME. The first of them is solved by considering the indicator matrix $\mathbf{U}$, so that the label fitness term forgets about unlabeled data. The second problem, which was derived from the terms D_{ii} in the cost function, also does not occur, because the label smoothness term considered is much simpler.

FME and unseen data

A really positive advantage of FME, which distinguishes it from many label propagation methods, e.g., from LGC, is its capacity of dealing with unseen data. Apart from predicting the labels of the unlabeled data in vector $\mathbf{f}$, it can predict unseen data using the regression term in 2.4. Given the features of the unseen data $\mathbf{X}_{unseen}$, its scores would be:

$$\mathbf{f}_{unseen} = \mathbf{X}_{unseen}^T \mathbf{w} + \mathbf{1}b.$$

Relationship between LGC and FME

The cost function of LGC in 2.1 can be expressed similarly to the cost function of FME in 2.3:

$$q(\mathbf{F}) = \mathrm{tr}(\mathbf{F}^T \hat{\mathbf{L}} \mathbf{F}) + \mu\, \mathrm{tr}[(\mathbf{F}-\mathbf{Y})^T(\mathbf{F}-\mathbf{Y})].$$

2.1.4 Kernel Flexible Manifold Embedding

Kernel Flexible Manifold Embedding (KFME) [5] is the kernel version of FME and it was proposed to cope with highly non-linear data (when a linear regression has a poor performance). The cost function in this case is

$$g(\mathbf{F},\mathbf{V}) = \mathrm{tr}(\mathbf{F}^T\mathbf{L}\mathbf{F}) + \beta\, \mathrm{tr}[(\mathbf{F}-\mathbf{Y})^T\mathbf{U}(\mathbf{F}-\mathbf{Y})] + \alpha\, \mathrm{tr}(\mathbf{V}^T\mathbf{K}\mathbf{V}) + \lambda\, \mathrm{tr}[(\mathbf{K}\mathbf{V}-\mathbf{F})^T(\mathbf{K}\mathbf{V}-\mathbf{F})],$$

where $\mathbf{K}$ is the kernel matrix of the data $\mathbf{X}$, in which each element K_{ij} is the result of applying the kernel function to samples i and j. In our case, the kernel will be Gaussian. This means that

$$K_{ij} = e^{-\dfrac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2t_0\sigma^2}},$$

where σ^2 is a measure of the variability of the data or, more concretely, it is the mean of the squares of the distances between all pairs of samples. t_0 is the kernel parameter, which will take different values.

A closed-form solution can be found again by setting the derivatives of g with respect to $\mathbf{F}$ and $\mathbf{V}$ as 0 (for more details, see [14]). The solution is

$$\mathbf{V} = \mathbf{A}\mathbf{F}$$

$$\mathbf{F} = \beta[\beta\mathbf{U} + \mathbf{L} + \alpha\mathbf{A}^T\mathbf{K}\mathbf{A} + \lambda(\mathbf{K}\mathbf{A}-\mathbf{I})^T(\mathbf{K}\mathbf{A}-\mathbf{I})]^{-1}\mathbf{U}\mathbf{Y},$$

where $\mathbf{A} = \lambda/\alpha(\mathbf{I} + \lambda/\alpha\mathbf{K})^{-1}$.

Kernel Flexible Manifold Embedding for regression

The cost function can also be defined for vectors of continuous real-valued labels $\mathbf{f}$ and $\mathbf{y}$ as follows:

$$g(\mathbf{f},\mathbf{v}) = \mathbf{f}^T\mathbf{L}\mathbf{f} + \beta(\mathbf{f}-\mathbf{y})^T\mathbf{U}(\mathbf{f}-\mathbf{y}) + \alpha\mathbf{v}^T\mathbf{K}\mathbf{v} + \lambda(\mathbf{K}\mathbf{v}-\mathbf{f})^T(\mathbf{K}\mathbf{v}-\mathbf{f}), \qquad (2.5)$$

Interpretation of the cost function

The interpretation is analogous to FME. The label smoothnes and label fitness criteria are exactly the same, while the linear regression becomes a non-linear regression thanks to the kernel trick. So the term $(\mathbf{Kv} - \mathbf{f})^T(\mathbf{Kv} - \mathbf{f})$ controls the loss in the non-linear regression and $\mathbf{v}^T\mathbf{Kv}$ controls the complexity of the model.

KFME and unseen data

Similarly to FME, in KFME one can easily handle unseen data using the regression term in 2.5. To do so, given a set of unseen samples $\mathbf{x}_i^{unseen}|_{i=1}^m$, one has to build the kernel matrix of the unseen samples $\mathbf{K}_{unseen} \in \mathbb{R}^{m\times N}$, where the element (i, j) is the kernel function of the unseen sample i, $\mathbf{x}_i^{unseen}$, and the training sample j, $\mathbf{x}_j$. Then, the predicted labels would be

$$\mathbf{f}_{unseen} = \mathbf{K}_{unseen}\mathbf{v}.$$

2.1.5 Flexible Graph-based Semi-supervised Manifold Embedding

Given the data matrix $\mathbf{X}$, this algorithm [3] obtains a non-linear embedding by optimizing a cost function based on several criteria, very similarly to the methods we have already seen in this chapter. The main difference is that up to the moment we were obtaining a vector of predictions $\mathbf{f}$ or a matrix of classes $\mathbf{F}$ and now we have a feature matrix $\mathbf{Z} \in \mathbb{R}^{N\times N}$. The objective of this algorithm is to clusterize the data so that samples belonging to the same classes form a group and samples belonging to different classes are widely separated. However, since the cost function is not trivial at all, our starting point will be a very simplified case, where the learning is supervised and $\mathbf{Z}$ "lives" in $\mathbf{R}^{N\times 1}$ or, in other words, the projected data points have only one dimension. Therefore, in the next section we will be assuming that all the N samples are labeled.

Supervised case with a single dimension

We will start defining the margin of a sample i. This is the difference between the mean of the inter-class distances, which measure the distance between sample i and samples belonging to different classes, and the mean of the intra-class distances, which measure the distance between sample i and samples belonging to its same class. In order to clusterize the data as has been commented, it is desirable for margins to be as large as possible. Assuming the number of dimensions of the non-linear embedding is 1, or equivalently that the non-linear projection $\mathbf{z}$ is a vector in $\mathbb{R}^N$ (below we will generalize this to N dimensions), the margin of a sample i belonging to a class C_k is mathematically defined as

$$m(i) = \frac{1}{l - l_k}\sum_{j\notin C_k}(z_i - z_j)^2 - \frac{1}{l_k}\sum_{t\in C_k}(z_i - z_t)^2,$$

where l_k is the number of samples belonging to class C_k.

The first term corresponds to the inter-class distance and the second one to the intra-class distance. However, since we wish to consider all of the samples and not just a single one, we should maximize the mean margin of all of the samples $\overline{m}$.

$$\overline{m} = \frac{1}{l}\sum_{i=1}^{l} m(i). \tag{2.6}$$

We will now consider the weighted inter-class and the intra-class graphs G_w and G_b, where nodes are all the samples. The former will be an undirected graph in which and edge will exist between two nodes if and only if they belong to the same class, while the second one will be directed and each edge starting at a node i will end in a node j belonging to a different class. The weight matrices associated to these graphs will be

$$S_{ij}^{w} = \begin{cases} \dfrac{1}{l_k} & \text{if } i \in C_k \text{ and } j \in C_k \\ 0 & \text{if } i \in C_k \text{ and } j \notin C_k \end{cases}$$

$$S_{ij}^{b} = \begin{cases} 0 & \text{if } i \in C_k \text{ and } j \in C_k \\ \dfrac{1}{l - l_k} & \text{if } i \in C_k \text{ and } j \notin C_k \end{cases}$$

After some algebraic manipulation (for further detail see [3]), one can express $\overline{m}$ in equation 2.6 in terms of $\mathbf{S}^w$ and $\mathbf{S}^b$:

$$\overline{m} = \frac{2}{l}\mathbf{z}^T\mathbf{D}_l\mathbf{z} - \frac{1}{l}\mathbf{z}^T\mathbf{M}_l\mathbf{z},$$

where $\mathbf{D}_l = \mathbf{I} + \mathbf{D}^b$ ($\mathbf{D}^b$ is the diagonal matrix whose diagonal elements are the row sums of $\mathbf{S}^b$) and $\mathbf{M}_l = 3\mathbf{I} + \mathbf{D}^b + \mathbf{S}^b + \mathbf{S}^{bT} - 2\mathbf{S}^w$.

We would like to maximize the mean margin $\overline{m}$. However, there is no solution to this problem. Why does this happen? Reasoning by contradiction, if we assume $\mathbf{z}_0$ were a solution which maximizes the margin, that would mean $\frac{2}{l}\mathbf{z}_0^T\mathbf{D}_l\mathbf{z}_0 - \frac{1}{l}\mathbf{z}_0^T\mathbf{M}_l\mathbf{z}_0$ is as large as possible. However, if we consider a proportional vector $\mathbf{z}_\lambda = \lambda\mathbf{z}_0$ with $\lambda > 1$, then the margin associated to $\mathbf{z}_\lambda$ would be

$$\frac{2}{l}\mathbf{z}_\lambda^T\mathbf{D}_l\mathbf{z}_\lambda - \frac{1}{l}\mathbf{z}_\lambda^T\mathbf{M}_l\mathbf{z}_\lambda = \frac{2}{l}(\lambda\mathbf{z}_0)^T\mathbf{D}_l(\lambda\mathbf{z}_0) - \frac{1}{l}(\lambda\mathbf{z}_0)^T\mathbf{M}_l(\lambda\mathbf{z}_0) = \lambda^2\left(\frac{2}{l}\mathbf{z}_0^T\mathbf{D}_l\mathbf{z}_0 - \frac{1}{l}\mathbf{z}_0^T\mathbf{M}_l\mathbf{z}_0\right),$$

which is λ^2 times the margin associated to $\mathbf{z}_0$ and, hence, it is larger. Thus, we have reached an absurd because we were assuming $\mathbf{z}_0$ had the largest margin.

This reasoning leads us to a clear conclusion: the search space is too vast to find only one solution and, in addition, $\mathbf{z}_0$ and $\mathbf{z}_\lambda$ are equivalent representations of the data, since they are proportional, so we should really get rid of the repeated solutions. A way of restricting the search space and avoiding this problem consists in imposing

$$\mathbf{z}^T\mathbf{D}_l\mathbf{z} = 1.$$

Then, our maximization problem becomes a minimization one with an associated restriction:

$$\mathbf{z} = \arg\min_{\mathbf{z}} \mathbf{z}^T\mathbf{M}_l\mathbf{z} \quad \text{s.t.} \quad \mathbf{z}^T\mathbf{D}_l\mathbf{z} = 1.$$

Generalization to N dimensions

To generalize the problem above, the first step is to use more dimensions instead of only 1 and making use of the unlabeled data, which will be done as with the previous methods, using the label smoothness criterion. So the minimization problem has to take into account two criteria. Firstly, the label smoothness in various dimensions,

$$\arg\min_{\mathbf{Z}} \operatorname{tr}(\mathbf{Z}^T\mathbf{L}\mathbf{Z}),$$

and, secondly, the maximization of the margin in a semi-supervised setting:

$$\arg\min_{\mathbf{Z}} \operatorname{tr}(\mathbf{Z}^T\tilde{\mathbf{M}}_l\mathbf{Z}) \quad \text{s.t.} \quad \mathbf{Z}^T\tilde{\mathbf{D}}_l\mathbf{Z} = \mathbf{I}.$$

In this context, $\tilde{\mathbf{M}}_l \in \mathbb{R}^{N\times N}$ and $\tilde{\mathbf{D}}_l \in \mathbb{R}^{N\times N}$ are the augmented forms of $\mathbf{M}_l \in \mathbb{R}^{l\times l}$ and $\mathbf{D}_l \in \mathbb{R}^{l\times l}$, that is

$$\tilde{\mathbf{M}}_l = \begin{pmatrix} \mathbf{M}_l & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \quad \text{and} \quad \tilde{\mathbf{D}}_l = \begin{pmatrix} \mathbf{D}_l & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

The zeros are due to the fact that there is no margin to be calculated for unlabeled samples. In addition, a regression term is added to the cost function, so that the non-linear projection $\mathbf{Z}$ is as close as possible to a linear embedding. This permits treating unseen samples, like in FME or KFME, by mapping them to the linear embedding. Considering all these factors the cost function becomes

$$e(\mathbf{Z},\mathbf{W},\mathbf{b}) = \operatorname{tr}(\mathbf{Z}^T\mathbf{L}\mathbf{Z}) + \lambda \operatorname{tr}(\mathbf{Z}^T\tilde{\mathbf{M}}_l\mathbf{Z}) + \mu(||\mathbf{W}||^2 + \gamma||\mathbf{X}^T\mathbf{W} + \mathbf{1}\mathbf{b}^T - \mathbf{Z}||^2)$$

$$\text{s.t.} \quad \mathbf{Z}^T\tilde{\mathbf{D}}_l\mathbf{Z} = \mathbf{I}.$$

$||W||^2$ is, again, a regularization term controlling the overfitting. The solution can be found by generalized eigenvalue decomposition.

2.1.6 Similarity graphs

Considering that the key to learning from unlabeled data is the similarity between samples, observing how the choice of the similarity matrix affects the results seems to be important. For this sake, we have tried three types of similarity matrices based on feature and score similarities between samples. It is important to recall that a similarity is a value ranging from 0 (when samples are unalike) to 1 (when they are alike).

Feature similarities

Given two instances i and j, let $\mathbf{x}_i$ and $\mathbf{x}_j$ be their respective features. Then, two different similarities are considered. Firstly, the *Gaussian similarity* is given by

$$s_{ij}^{gauss} = e^{-\dfrac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}},$$

were σ^2 is the mean of $||\mathbf{x}_i - \mathbf{x}_j||^2$ for $i, j = 1, \cdots, n$. According to the formula, samples are considered to be similar when their features are close in the Euclidean distance.

Secondly, the *cosine similarity* is defined as follows:

$$s_{ij}^{cos} = \frac{\dfrac{\mathbf{x}_i \mathbf{x}_j}{||\mathbf{x}_i|| ||\mathbf{x}_j||} + 1}{2}.$$

This similarity corresponds to the cosine between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, rescaled so that it lies on the interval [0, 1], instead of [-1, 1]. So two samples are considered to be more similar when the angle formed by their respective vectors becomes narrower. The rescaling to avoid negative similarities is not strictly necessary for all the algorithms. In fact, it is only needed with LGC, because equation 2.1 uses the square roots of the row sums of the similarity matrix, $\sqrt{D_{ii}}$, so these sums cannot be negative. However, for simplicity, the rescaled cosine similarity is always used.

It is interesting to consider both similarities, since they measure different things. Even if using hundreds of dimensions makes the geometric sense vanish, how both similarities differ can be easily noticed, since the cosine similarity does not care about the norm of the features, while vectors with very different norm values will always be considered unalike by the Euclidean or Gaussian similarity.

Score similarity

Given two labeled samples i and j, and their respective labels y_i and y_j, they will be similar if the difference between the scores does not exceed a certain threshold $\delta > 0$. Mathematically this can be stated as

$$s_{ij}^{score} = \begin{cases} 1 & \text{if } |y_i - y_j| < \delta \\ 0 & \text{otherwise} \end{cases}$$

This binary similarity is the simplest one which could have been considered and, even though it will not be studied here, it could be good idea to analyse more complex score similarities.

Construction of the similarity matrix

Three different types of similarity matrices are built in this study: the similarity matrix based only on Gaussian feature similarity and matrices which are based on Gaussian or cosine similarity and score similarity. All of them will depend on the idea of the k nearest neighbors.

The process is the following:

1. The feature similarities of each pair of samples are computed.

2. For each of the samples the k nearest neighbors are considered ($k = 10$ throughout all the work), while the farthest samples are considered to have 0 similarity. This can be understood as a directed graph where each node is a sample and the weight of each edge is the similarity between samples. The edges corresponding to the farthest neighbors are removed. An initial similarity matrix $\mathbf{S}^{initial}$ is built based on the directed graph.

3. Since the similarity matrix has to be symmetric, a new similarity matrix $\mathbf{S}$ is built according to the operation:

$$S_{ij} = \max(S_{ij}^{initial}, S_{ji}^{initial}),$$

which is the same as converting our directed graph into an undirected one by considering all the edges bidirectional instead of unidirectional.

When a feature similarity and the score similarity are considered, there is an additional step between step 1 and step 2. Before selecting the k nearest neighbors, all the pairs whose score difference exceeds a given threshold δ are removed. Then, in step 2, for each sample, the k nearest neighbors are selected among the remaining samples (whose score difference is smaller than δ).

2.2 Supervised algorithms

In this section, three classical regression algorithms are presented, which will be compared with the semi-supervised algorithms in the next chapter to check whether the semi-supervised approach provides an improvement.

2.2.1 Mathematical notation of the supervised algorithms

The difference between the semi-supervised algorithms and the supervised algorithms is that the latter do not use the unlabeled data in training. This affects the notation, where we will have a clear separation between the training data and the test data. N will be the number of training data and M the number of test data. $\mathbf{X}_{train} \in \mathbb{R}^{N\times D}$ and $\mathbf{y}_{train} \in \mathbb{R}^{N}$ will note the training data matrix, where samples are represented as columns, and the labels of the training data, respectively. And the test data and labels will be $\mathbf{X}_{test} \in \mathbb{R}^{D\times M}$ and $\mathbf{y}_{test} \in \mathbb{R}^{M}$.

2.2.2 1-Nearest Neighbor

k-nearest neighbors (KNN) is a lazy learning algorithm, which means that it does not have a real training phase, since it simply memorizes the training data. On the other hand, the prediction step is when the algorithm is actually working. In order to evaluate a new point, it finds the k nearest training points and the label assigned to this new point only depends on the labels of the k nearest neighbors. In our case, the number k of neighbors will be 1, so, when evaluating a new point, the algorithm will just find its nearest neighbor and copy its label.

This algorithm is fundamentally based on the concept of nearness; however, this can be relative, depending on the type of distance considered. In this work, three basic distances are used: the Euclidean distance, the Minkowski distance and the cosine distance. Given two real row vectors of size n, $\mathbf{x} = [x_1, \cdots, x_n]$ and $\mathbf{y} = [y_1, \cdots, y_n]$, each of the distances is defined as follows:

- Euclidean distance:
$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^T}.$$
- Minkowski distance of exponent $p \in \mathbb{N}$:
$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}.$$
  The Euclidean distance is a particular case of the Minkowski distance, when $p = 2$.
- Cosine distance:
$$d(\mathbf{x}, \mathbf{y}) = 1 - \frac{\mathbf{x}\mathbf{y}^T}{\sqrt{(\mathbf{x}\mathbf{x}^T)(\mathbf{y}\mathbf{y}^T)}}.$$
  The cosine distance derives from the cosine similarity and it ranges from 0 (when $\mathbf{x} = \lambda\mathbf{y}$ with $\lambda > 0$) to 2 (when $\mathbf{x} = \lambda\mathbf{y}$ with $\lambda < 0$). It is 1 when the vectors pointing from the origin of the coordinate system to the points $\mathbf{x}$ and $\mathbf{y}$ respectively are orthogonal. So this distance grows with the angle between the vectors $\mathbf{x}$ and $\mathbf{y}$.

2.2.3 Ridge Regression

Ridge regression (RR), named by Hoerl and Kennard [8], is a sophistication of the least squares method by adding a regularization term. It aims at minimizing the cost function
$$e(\mathbf{w}) = ||\mathbf{w}^T\mathbf{X}_{train} - \mathbf{y}_{train}|| + \lambda||\mathbf{w}||^2,$$
where $\mathbf{X}_{train}$ usually is the centered and scaled training data, that is, each column has mean 0 and standard deviation 1. The term $||\mathbf{w}||^2$ controls the complexity of the model or, in other words, it avoids overfitting.

2.2.4 Support Vector Regression

ϵ-insensitive Suport Vector Regression (ϵ-SVR) [18, 20] is a regression algorithm in which an instance i with a label y_i is considered to be correctly predicted by the machine if its prediction lies on the interval $[y_i - \epsilon, y_i + \epsilon]$. The regression can be linear or non-linear (using the kernel trick). Firstly we will describe the linear case. Given an instance $\mathbf{x}$, we aim at obtaining a regression function $f(\mathbf{x})$ of the form

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b, \qquad \text{with } \mathbf{w}, \mathbf{x} \in \mathbb{R}^D \text{ and } b \in \mathbb{R}.$$

Similarly to Ridge Regression, ϵ-SVR is posed as a minimization problem in which two terms have to be taken into account. On one hand, the regression fitness (taking into account that predictions in the interval $[y_i - \epsilon, y_i + \epsilon]$ have 0 error) and, on the other, the complexity of the model given by $||\mathbf{w}||^2$. These conditions can be comprised in an optimization problem stated as follows:

$$\min \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } \begin{cases} y_i - (\mathbf{w}^T\mathbf{x}_i + b) \leq \epsilon \\ \mathbf{w}^T\mathbf{x}_i + b - y_i \leq \epsilon \end{cases}$$

The cost function corresponds to the complexity of the model and the two restrictions force the samples to be predicted correctly. This statement of the problem, though, is too rigid, since it does not permit any of the training samples to be predicted outside the ϵ tube. This can be tackled by adding slack variables ξ_i and ξ_i^* permitting some deviations in the model:

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{subject to } \begin{cases} y_i - (\mathbf{w}^T\mathbf{x}_i + b) \leq \epsilon + \xi_i \\ \mathbf{w}^T\mathbf{x}_i + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \tag{2.7}$$

C is the constant which controls the trade-off between the complexity and the flexibility of the model. A high C implies that ξ_i and ξ_i^* tend to be lower, leading to a rigid model, where every sample has to be accurately classified and, in addition $||\mathbf{w}||^2$ is not hardly penalized. Therefore, the model tends to overfitting. A low C means just the opposite, tending the model to underfitting. So a correct choice of the parameter C is a vital decision in order to get a good performance of the algorithm.

In figure 2.2 one can see how the regression error of ϵ-SVR, ζ, is computed, which is always lower than the real regression error, because predictions are considered to be correct within the given tolerance ϵ. So a real regression error of ϵ costs 0 and, for higher errors, ζ is obtained by substracting a quantity of ϵ to the real error.

Kernel Support Vector Regression

Kernel Support Vector Regression arises to overcome the limitations of the linear regression. Instead of considering the raw features of a sample $\mathbf{x}$, a non-linear mapping of the features $\phi(\mathbf{x})$, is used before performing the linear regression. Equation 2.7

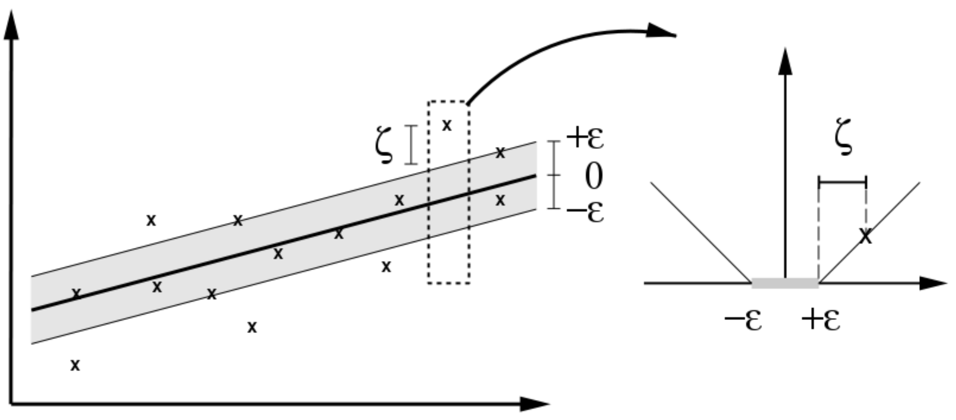


Figure 2.2: The soft margin loss, as shown in [19].

would be expressed as

$$\min \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{subject to} \begin{cases} y_i - (\mathbf{w}^T\phi(\mathbf{x}_i) + b) \leq \epsilon + \xi_i \\ \mathbf{w}^T\phi(\mathbf{x}_i) + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

The solution can be found thanks to the dual optimization problem, where the use of an explicit non-linear mapping ϕ is avoided and, instead, a kernel function K implicitly maps the data non-linearly. Since this algorithm is not the main point of the work, we will skip the technical explanation regarding this part. So, for further detail, the reader can see the references [18, 20].

Chapter 3

Empirical study

3.1 General methodology

As mentioned in section 1.3.1, two datasets are used in our study: SCUT-FBP and M^{2}B. Most of the experiments have been carried out in SCUT-FBP, since it contains higher quality photos and, supposedly, it should make the learning easier. Labels in SCUT-FBP lie on the interval (1, 5), whereas in M^{2}B they lie on [1, 10]. In both cases, labels are scaled so that they lie on [0.1, 1], dividing by 5 in the first case and by 10 in the second.

Several of the algorithms used in this study have a number of parameters to be fixed. In all of these cases a grid search is conducted, that is, given a finite set of possible values for each parameter, all of the combinations are tried and the one giving the best error is selected.

All the matlab code used in these experiments is replicable downloading it from the following link: `https://github.com/AnneED/aesthetic-analysis.git`.

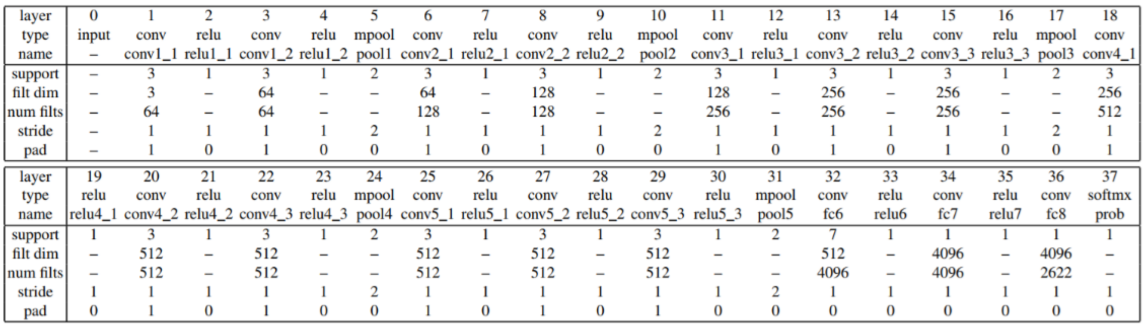
3.1.1 Face features

Recent studies have proved that extracting features using a pretrained Convolutional Neural Network (CNN) can provide more than satisfactory results in computer vision [17], often improving methods using other types of features and even if they are used in a task which is not exactly the one the network was trained to accomplish. Therefore, we are adopting this approach for our study, using the VGG-face network [15], which was trained for face identification and is available for matlab[1]. Figure 3.1 shows its architecture.

The features are extracted from layer 7, right before the last fully-connected layer (fc8 in the diagram). Before propagating an image through VGG-face, it must be resized and the average of the network has to be substracted.

After extracting the features using the CNN, they are organized in a matrix, where each column represents the 4096 features of an image. The matrix is normalized using

[1]The network can be downloaded from `http://www.vlfeat.org/matconvnet/pretrained/`.



layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
type	input	conv	relu	conv	relu	mpool	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv
name	–	conv1_1	relu1_1	conv1_2	relu1_2	pool1	conv2_1	relu2_1	conv2_2	relu2_2	pool2	conv3_1	relu3_1	conv3_2	relu3_2	conv3_3	relu3_3	pool3	conv4_1
support	–	3	1	3	1	2	3	1	3	1	2	3	1	3	1	3	1	2	3
filt dim	–	3	–	64	–	–	64	–	128	–	–	128	–	256	–	256	–	–	256
num filts	–	64	–	64	–	–	128	–	128	–	–	256	–	256	–	256	–	–	512
stride	–	1	1	1	1	2	1	1	1	1	2	1	1	1	1	1	1	2	1
pad	–	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1

layer	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
type	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	softmx
name	relu4_1	conv4_2	relu4_2	conv4_3	relu4_3	pool4	conv5_1	relu5_1	conv5_2	relu5_2	conv5_3	relu5_3	pool5	fc6	relu6	fc7	relu7	fc8	prob
support	1	3	1	3	1	2	3	1	3	1	3	1	2	7	1	1	1	1	1
filt dim	–	512	–	512	–	–	512	–	512	–	512	–	–	512	–	4096	–	4096	–
num filts	–	512	–	512	–	–	512	–	512	–	512	–	–	4096	–	4096	–	2622	–
stride	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
pad	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0

Figure 3.1: Architecture of VGG-face from [15].

L2 normalization (each column is divided by its Euclidean norm) and, then, the matrix is reduced from 4096 to 200 features using Principal Component Analysis (PCA).

3.1.2 Model evaluation

In order to have an idea of the effect of different data sizes during the training phase, which can be particularly interesting considering that we are working in a semi-supervised setting because there exist limited labeled data for this problem, many data size configurations are tested. In most of the experiments the labeled data constitute the 50%, the 70% or the 90% of all the data. On the other hand, when using semi-supervised algorithms the rest of the data is the unlabeled part, which is used in the training, as well as in the test phase, except for the experiment shown in table 3.10, in which the effect of testing in unseen data is studied.

To have a good estimation of the error of the models, given the training/test percentages, each model is trained 10 times with 10 different splits of the data, sampled using stratification so that the label distributions are roughly equal in both the training and the test sets. The error is estimated as the mean of the errors of the 10 splits. Since our labels are continuous numbers, five "artificial" classes are created to carry out the stratification easily, as shown in table 3.1. In figures 3.2, 3.3 and 3.4 histograms of the classes are presented. These procedures will be maintained in each of the experiments unless otherwise indicated. Each model is evaluated using the performance metrics described in the next section.

3.2 Performance metrics

In order to measure the performance of the different algorithms, four metrics are taken into account: the mean absolute error (MAE), the root mean square error (RMSE), the Pearson correlation coefficient (PC) and the ϵ-error.

Class	Interval in SCUT-FBP	Interval in M^2B
1	[1,2)	[1,3)
2	[2, 3)	[3, 5)
3	[3, 4)	[5, 7)
4	[4, 4.5)	[7, 8)
5	[4.5, 5)	[8, 10)

Table 3.1: Transformation of the labels from real numbers to discrete classes.

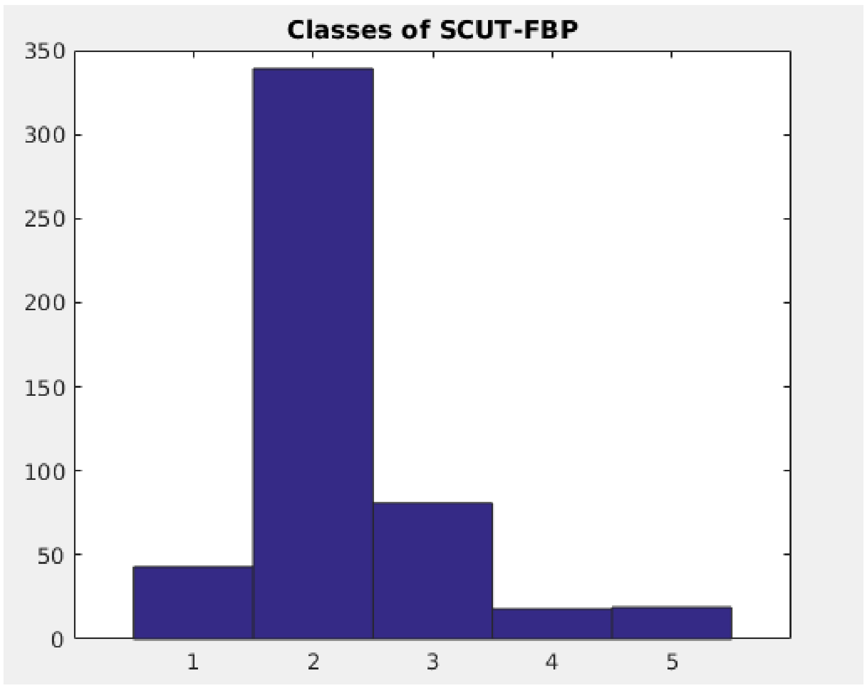


Figure 3.2: Histogram of classes in SCUT-FBP dataset.

Mean absolute error

Let $y_1, y_2, \cdots, y_n$ be the ground truth labels and $f_1, f_2, \cdots, f_n$ the estimated labels, then the mean absolute error of the prediction is given by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - f_i|.$$

Root mean square error

Let $y_1, y_2, \cdots, y_n$ be the ground truth labels and $f_1, f_2, \cdots f_n$ the estimated labels, then the root mean square error of the prediction is given by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - f_i)^2}.$$

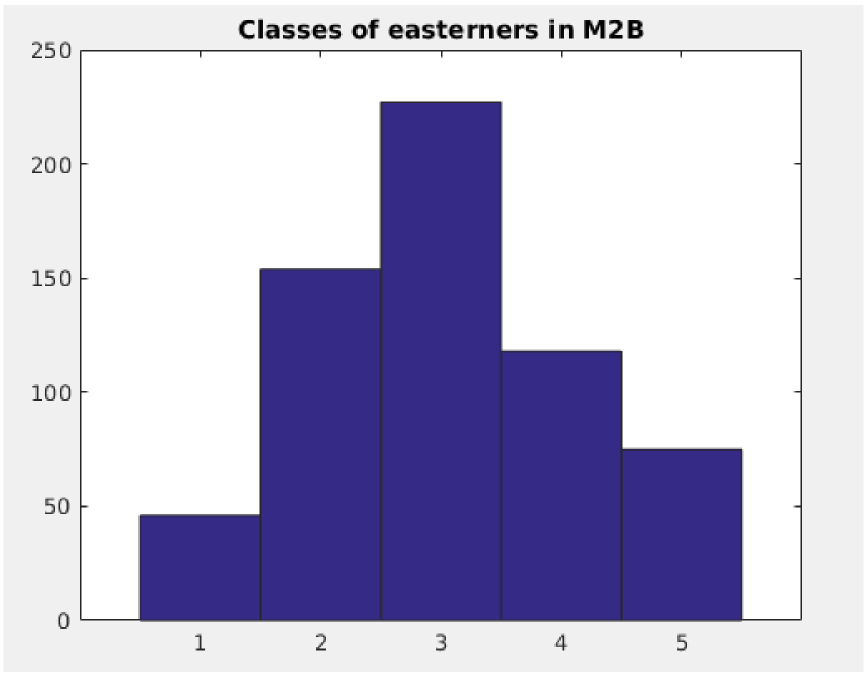


Figure 3.3: Histogram of the classes of the easterners in M^{2}B dataset.

Pearson correlation coefficient

Let $y_1, y_2, \cdots, y_n$ be the ground truth labels and $f_1, f_2, \cdots f_n$ the estimated labels, then the Pearson's correlation coefficient measures the linear correlation between the ground truth and the estimated labels and is given by:

$$PC = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(f_i - \bar{f})}{\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}\sqrt{\sum_{i=1}^{n}(f_i - \bar{f})}},$$

where $\bar{f}$ and $\bar{y}$ are the means of the estimated labels and the ground truth labels respectively.

The PC is a number in the interval [-1, 1]. It measures to what extent the points (y_i, f_i), with $i = 1, \cdots, n$, fit in their regression line. A PC around 1 or -1 means that there is a high linear correlation, where the slope of the regression line is positive, if the PC is 1, or negative, if the PC is -1. On the other hand, a PC around 0 means that there exists no linear correlation.

In our particular case, a PC as close to 1 as possible is desirable, since a perfect prediction would be one in which $f_i = y_i$, in other words, the pairs (y_i, f_i) lie on the line $y = x$.

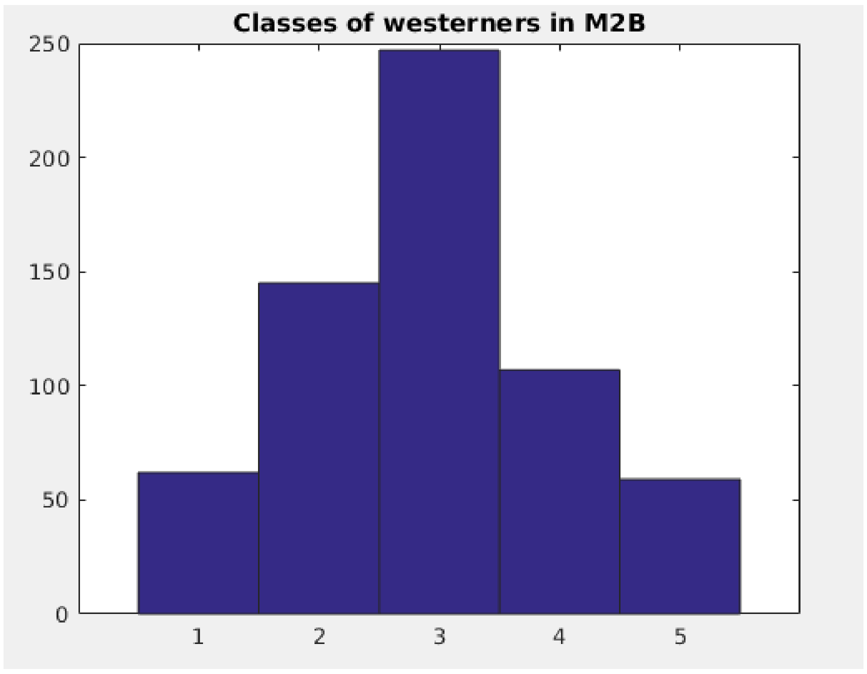


Figure 3.4: Histogram of the classes of the westerners in M^{2}B dataset.

ϵ-error

Let $y_1, y_2, \cdots, y_n$ be the ground truth labels and $f_1, f_2, \cdots, f_n$ the estimated labels, then the ϵ-error of the prediction is given by:

$$\epsilon\text{-error} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - e^{-\dfrac{(y_i - f_i)^2}{2\sigma_i^2}}\right),$$

where σ_i is the standard deviation of the ratings of all the raters of image i.

This error lies on the interval $[0, 1)$. Unlike the MAE and the RMSE, ϵ- error takes into account the ratings of all the raters of each image. Since beauty is a relative concept, raters may not agree when judging it. What is more, one can consider two types of faces: the ones which are found attractive by certain people and which are not by others and the faces which are judged mostly equally by all the raters. In other words, the ratings of the different raters of a face of the first group will have a higher standard deviation than the ratings of a face of the second group. So, if the predictor does not exactly predict the mean of the ratings of a face with a high standard deviation, the error is considered not to be so relevant, since there is no human agreement about the rating of that particular face. This is why in the formula the square difference between the real score and the estimated score $(y_i - f_i)^2$ is divided by σ_i^2.

3.3 Results on SCUT-FBP dataset

3.3.1 The baseline

Table 3.2 shows the performance of an algorithm which, regardless of the face which it is presented, the score returned by it will be the average of the labels of the training images. This will be our starting point. While the PC could not be worse, the mean average error is not so high, since the labels follow a normal distribution and most of the images are gathered around the average.

3.3.2 Local and Global Consistency

As explained in section 2.1.2, LGC has a parameter μ which has to be adjusted. In all the experiments related to LGC, its value is selected among the following so that the MAE is minimized: $\{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$.

As hypothesized in the previous chapter in section 2.1.2, the "label" given to unlabeled images, that is, the value of y_i when i corresponds to an unlabeled instance, seems to affect the performance of this algorithm when it comes down to regression. Table 3.3 confirms this assumption. It shows the performance of the algorithm when unlabeled instances are "labeled" as 0 or as the mean of the labeled instances. The similarity matrix used to compute the Laplacian is the simplest one, based only on Gaussian feature similarity. Being the difference so considerable, we are setting y_i (when i corresponds to an unlabeled instance) as the average in the rest of the experiments regarding LGC.

Table 3.4 shows the results of applying LGC with different training/test sizes and Laplacians (where $\delta = 0.1$, when computing the similarity matrices) and table 3.5 shows the effect of similarity graph normalization in LGC when the Laplacian is only based on feature similarity. Even though the improvement is apparent, which was forseeable considering the discussion in section 2.1.2, it is not as marked as could be expected.

3.3.3 Flexible Manifold Embedding

As mentioned in section 2.1.3, FME has three parameters, β, μ and γ to be adjusted. All of them are selected among $\{10^{-9}, 10^{-6}, 10^{-3}, 1, 10^{3}, 10^{6}, 10^{9}\}$ in order to optimize the MAE.

Table 3.6 shows the performance of FME considering different data sizes and Laplacians (where $\delta = 0.1$, when computing the similarity matrices), proving that, in this case, the results are independent of the Laplacian.

3.3.4 Kernel Flexible Manifold Embedding

As mentioned in section 2.1.4, KFME has three parameters to be adjusted in its cost function, β, α and λ, as well as a fourth parameter t_0, which controls the width of the kernel function. β is selected among $\{0.1, 1, 10, 100, 1000, 10000\}$, α among $\{0.0001, 0.001, 0.01, 0.1, 1, 10\}$, λ among $\{1, 10, 50, 100, 1000\}$ and t_0 among $\{1/8, 1/4, 1/2, 1, 2, 4, 8\}$.

Unexpectedly, being KFME the kernelized version of FME, the first time KFME was used in this study, the MAE was worse than using the simpler algorithm. More surprisingly, when the PC was optimized in KFME, it was around 0.85, slightly outperforming FME. Given this information, we should remember that the PC measures the linear correlation between two variables or, in other words, how well a regression line between these two variables would fit. A perfect performance would be one in which, for each test sample i, the predicted label is exactly the real label, that is $y_i = f_i$, so the MAE would be 0 and the PC 1, because, if we plot each pair (f_i, y_i), it lies on the line $y = x$. However, having a poor MAE and such a good PC means that the pairs (f_i, y_i) almost lie on a perfect line, but since it cannot be $y = x$, it has to be a distorted line $y = ax + b$. In fact, after plotting this in our specific problem, the result was the corresponding to figure 3.5.

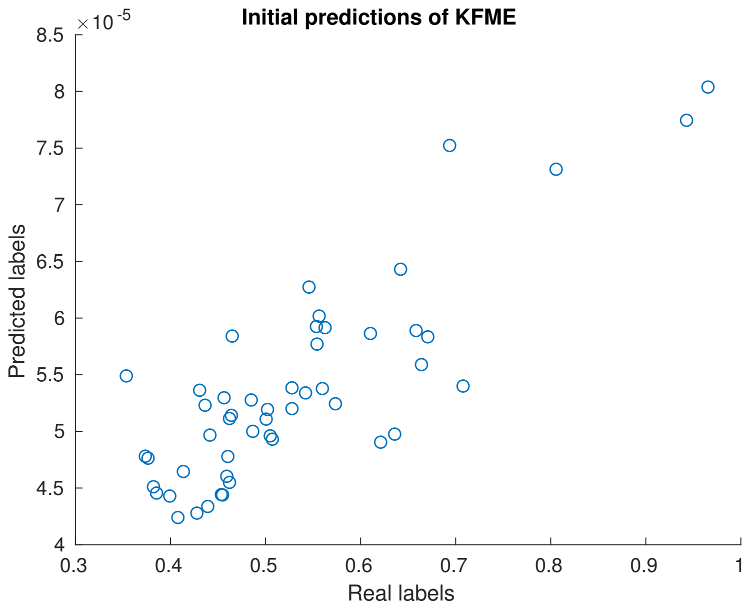


Figure 3.5: Initial predictions of KFME.

The way of fixing this problem was just rescaling the output $\mathbf{f}$ of the algorithm, according to the formula:

$$\mathbf{f}_{scaled} = \min(\mathbf{y}_{train}) + (\mathbf{f} - \min(\mathbf{y}_{train}))\frac{\max(\mathbf{y}_{train}) - \min(\mathbf{y}_{train})}{\max(\mathbf{f}) - \min(\mathbf{f})}. \tag{3.1}$$

Table 3.7 shows the improvement when scaling the output of the algorithm $\mathbf{f}$. The possible reasons of this need to scale $\mathbf{f}$ will be discussed in section 4.1. In addition, after observing the great difference of scaling the output in KFME, the scaling was also used in FME, but no improvement was observed in this case.

Table 3.8 shows the performance of KFME using different training/test proportions and Laplacians based on different similarities (where $\delta = 0.1$, when computing the similarity matrices). It seems that using score similarity as well as feature similarity slightly improves the performance, but the difference is not so marked. On the other hand, table 3.9 shows the results of varying the value of δ when computing the Laplacian based on Gaussian feature similarity combined with score similarity. Although the optimal value seems to be around 0.2, the value of δ does not seem to affect strongly the results.

An experiment on unseen data

It could be argued that our study, as yet, has a major flaw, namely, that the unlabeled data is used both in training and in testing and, therefore, that nobody can ensure its correct performance on new data. This experiment is an attempt to address the unseen data case. We are considering the following partition of the data (sampled with stratification): 200 images will form the labeled data, 200 will be the unlabeled data and the remaining 100 will be the test data. The algorithm will be trained on the 400 labeled and unlabeled instances (the 80% of the data) and it will be tested on the remaining 100 (the 20% of the data), as explained in section 2.1.4, using the linear regression on the kernel matrix associated to unseen samples.

Table 3.10 shows the best of the models considering all the possible combinations of parameters. In this case, only one data partition has been considered, instead of the usual 10 splits, in order to simplify the task. In addition, when scaling the images as described in equation 3.1, instead of taking the maximum and minimum labels of the labeled training data, since there are only 200 labeled instances, the maximum and the minimum of all the data are used.

This experiment shows that KFME can work correctly for unseen images.

3.3.5 Flexible Semi-supervised Embedding

As opposed to the other semi-supervised algorithms, which aim at classification or direct regression, here we will see the results of using the embedding described in section 2.1.5, combined with a supervised algorithm. Firstly, we will be using linear ϵ-SVR and then 1-NN. The algorithm to obtain the embedded features, which we will be naming "zwb features", has three parameters, λ, μ and γ, that have to be adjusted. We are choosing all of them among $\{10^{-9}, 10^{-6}, 10^{-3}, 1, 10^{3}, 10^{6}, 10^{9}\}$. In addition, the number of dimensions of the output $\mathbf{Z}$ used to represent the data has to be selected, since, similarly to PCA, the matrix $\mathbf{Z}$ can be cropped so that the data has fewer dimensions. The possible number of dimensions will be ranging from 10 to 500, with a step of 10, and the parameter t_0 used in the kernel function, which is the same as in KFME, has to be selected among $\{1/8, 1/4, 1/2, 1, 2, 4, 8\}$.

The training/test configuration in ϵ-SVR and in 1-NN will be the same as the labeled/unlabeled configuration when using the semi-supervised embedding.

In ϵ-SVR we will be adjusting the parameters C and ϵ. Since there are 6 or 7 parameters to optimize (5 or 6 in the non-linear embedding, depending on whether we are using the kernel version, and 2 in ϵ-SVR) and this can be very time-consuming, firstly the parameters in the embedding are optimized using the default parameters for SVR, and then the parameters in ϵ-SVR are optimized. C and ϵ are selected among $\{5, 7.5, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5\}$ and $\{0, 0.005, 0.01, 0.02, 0.03, 0.04\}$, respectively, after seeing, using a wider search, that the optimal values were around 20 and 0.01. Table 3.11 shows the comparison of applying ϵ-SVR on the raw features or on the ones obtained with the embedding. Judging by the results, it is not clear that using the non-linear transformation has a positive effect. However, there is a slight improvement for the 90%/10% partition and for the non-kernelized variant. Our future work is to integreate the score similarity in the margin used by the non-linear projection method.

Experiment on discrete classes

On the other hand, table 3.12 shows a clear improvement when using the non-linear transformation (if we look at the Euclidean distance), possibly implying that, even though this embedding works properly on discrete classes, maybe it is not so suitable for regression problems. In this second case, 3 discrete classes are created: most attractive, average beauty and least attractive. To create these classes only 400 images are used and the intermediate classes are discarded. In this case, only 1 split of the data has been considered, instead of repeating the experiment 10 times. Table 3.13 shows the normalized score intervals on which lie each of the classes.

Table 3.12 shows the correct prediction rates of class 1, class 2, class 3 and all of the samples. When zwb features are used, the parameters are optimized in order to obtain the best total correct prediction rate. When using the Euclidean distance, the improvement in the prediction is remarkable, whereas the cosine distance does not imply a real improvement. However, this has much sense. The graph-based embedding method aims at clusterizing the samples depending on their classes and, for this purpose, it uses the Euclidean distance, so this is the reason for the 1-NN Euclidean classifier to work better.

3.3.6 Supervised algorithms

In this section, we will report the performances of three supervised algorithms, namely, 1-Nearest Neighbor, Ridge Regression and ϵ-insensitive Support Vector Regression, in order to compare them with the semi-supervised case.

Table 3.14 shows the results of 1-NN on the raw features of SCUT-FBP with the Euclidean distance, the cosine distance and the Minkowski distance with the exponent $p = 1$. The best results are achieved by the Euclidean distance with a training/test configuration of 90%/10%.

Table 3.15 shows the results of applying Ridge Regression on SCUT-FBP dataset. As explained in section 2.2.3, Ridge Regression has a parameter μ that has to be fixed. The different parameters tried for this purpose have been {0.0001, 0.001, 0.01, 0.1, 1, 10, 50, 100, 250, 500, 1000, 5000, 10000}.

Table 3.16 shows the results of applying ϵ-insensitive SVR with a linear and a Gaussian kernel. In the first case, two parameters have to be adjusted, C and ϵ, which have been selected among {0.00001, 0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 2, 3, 4, 5, 10, 20, 3} and {0, 0.001, 0.0025, 0.005, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.15}. In the second case there is an additional kernel parameter, which has been chosen among $\{1/2^5, 1/2^4, 1/8, 1/4, 1/2, 1, 2, 4, 8, 2^4, 2^5\}$.

3.3.7 Method comparison: Regression Error Characteristic curve

A way of plotting and comparing a number of regression methods is using the Regression Error Characteristic (REC) curve [2]. Given a regression method and a test sample i labeled as y_i and predicted as f_i, the regression error of this test sample can be either its squared error, $(y_i - f_i)^2$, or its absolute error, $|y_i - f_i|$. We will be using the latter. We can fix a tolerance value ϵ and count the number of images whose error is smaller than the tolerance ϵ. This is exactly what the REC curve plots. The X axis will be a range of tolerances and the Y axis will be the proportion of images whose error is smaller than this tolerance.

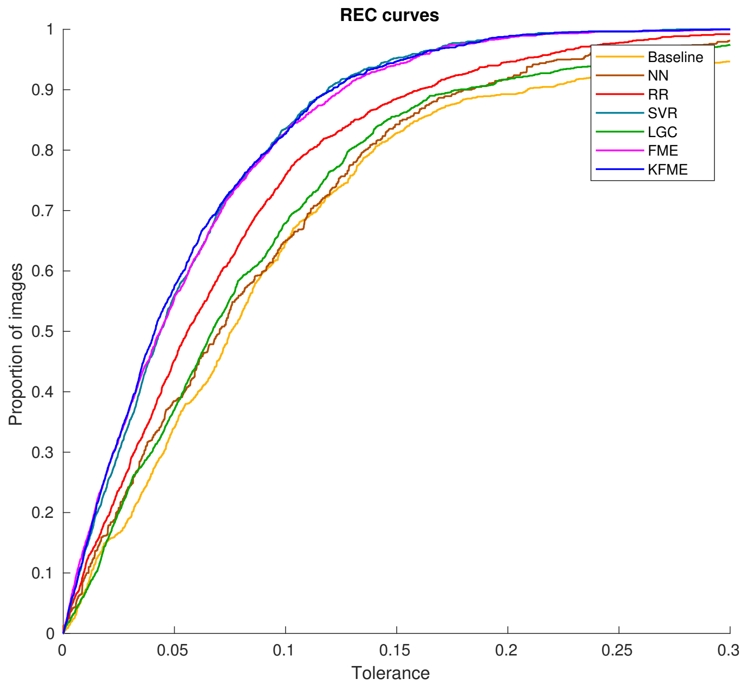


Figure 3.6: REC curves of different methods. The steeper a curve is the better is the performance of the algorithm.

Figure 3.6 shows the REC curve of the best models of the mean predictor, 1-NN, Ridge Regression, Gaussian ϵ-SVR, LGC, FME and KFME in the case where 70% of the data constitute the labeled part. A steep curve indicates that a greater number of images has smaller errors and the area over the curve is an estimate of the error. Hence, a smaller area indicates a better performance.

Having a quick look at the figure, it seems clear that KFME (in dark blue) outperforms all the others. For a more detailed comparison, see table 3.17, which summarizes the best results obtained by each algorithm with a fixed data configuration of 90%/10%.

3.4 Results on M^{2}B dataset

In this section, we will see the results on M^{2}B dataset, with a configuration of 50% of the samples as labeled data and the other 50% of the samples as the unlabeled/test data. As stated in the beginning of the chapter, all the experiments are carried out doing 10 stratified splits of the data, except for table 3.23.

3.4.1 The baseline

Table 3.18 shows the results of an algorithm which returns always the average of the labels of the training images, regardless of the image which it is presented. The MAE and the RMSE are much higher than the baseline in SCUT-FBP (table 3.2). This is due to two facts: firstly, the normalized labels lie on a narrower interval in SCUT-FBP (the lowest label in SCUT-FBP is 0.2394, while on M^{2}B it is 0.1). In addition, even though in both datasets the beauty scores roughly follow a Gaussian distribution, in SCUT-FBP this is much sharper, implying that labels in M^{2}B do not gather as much around the mean.

3.4.2 Local and Global Consistency

Table 3.19 in M^{2}B is the equivalent to table 3.4 in SCUT-FBP, with the only difference that in this case the similarity matrix has been normalized after seeing that the results in SCUT-FBP improved thanks to normalization (table 3.5).

3.4.3 Flexible Manifold Embedding

Table 3.20 represents the results of using FME on M^{2}B under the same conditions as in 3.6. The results are independent of the type of similarity used to build the Laplacian matrix.

3.4.4 Kernel Flexible Manifold Embedding

Tables 3.21 and 3.22 are the corresponding to tables 3.8 and 3.9 in SCUT-FBP. It is not clear which type of similarity graph is the most convenient. In this case, the choice of δ in the similarity graph construction does not truely affect the results (the slight difference is only appreciated in the last decimal of the PC).

3.4.5 Flexible Semi-supervised Embedding

Tables 3.23 and 3.24 are the corresponding to tables 3.12 and 3.11 in SCUT-FBP. The three classes used in table 3.23 are described in table 3.25. In this experiment only the eastern samples have been taken into account. It seems that, contrary to what happened with SCUT-FBP, in both cases using the non-linear embedding improves the results.

3.5 Tables

Data size	MAE	RMSE	PC	ϵ-error
50% train / 50% test	0.1009	0.1351	0.0000	0.2586
70% train / 30% test	0.0979	0.1322	0.0000	0.2479
90% train / 10% test	0.0987	0.1309	0.0000	0.2557

Table 3.2: Results of the baseline.

Data size	Labels	MAE	RMSE	PC	ϵ-error
50% labeled 50% unlabeled	Zero Mean	0.2711 0.0940	0.3032 0.1230	0.1002 0.4520	0.7395 0.2417
70% labeled 30% unlabeled	Zero Mean	0.1711 0.0876	0.2137 0.1144	0.0976 0.5646	0.4388 0.2212
90% labeled 10% unlabeled	Zero Mean	0.1100 0.0850	0.1497 0.1097	0.1153 0.6223	0.2635 0.2134

Table 3.3: Difference between setting y_i as 0 or as the mean of the known labels when i corresponds to an unlabeled instance.

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% labeled 50% unlabeled	Gaussian	0.0940	0.1230	0.4520	0.2417
	Gaussian + score	0.0945	0.1211	0.4777	0.2481
	Cosine + score	0.0946	0.1212	0.4755	0.2483
70% labeled 30% unlabeled	Gaussian	0.0876	0.1144	0.5646	0.2212
	Gaussian + score	0.0889	0.1115	0.5934	0.2340
	Cosine + score	0.0889	0.1115	0.5917	0.2341
90% labeled 10% unlabeled	Gaussian	0.0850	0.1097	0.6223	0.2134
	Gaussian + score	0.0954	0.1144	0.6423	0.2686
	Cosine + score	0.0953	0.1144	0.6417	0.2684

Table 3.4: Results of LGC using different Laplacians.

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% labeled 50% unlabeled	Raw	0.0940	0.1230	0.4520	0.2417
	Normalized	0.0928	0.1217	0.5952	0.2375
70% labeled 30% unlabeled	Raw	0.0876	0.1144	0.5646	0.2212
	Normalized	0.0861	0.1132	0.6905	0.2169
90% labeled 10% unlabeled	Raw	0.0850	0.1097	0.6223	0.2134
	Normalized	0.0830	0.1076	0.7390	0.2090

Table 3.5: Effect of normalizing the similarity matrix in LGC.

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% labeled 50% unlabeled	Gaussian	0.0626	0.0814	0.8162	0.1421
	Gaussian + score	0.0626	0.0814	0.8162	0.1421
	Cosine + score	0.0626	0.0814	0.8162	0.1421
70% labeled 30% unlabeled	Gaussian	0.0563	0.0746	0.8396	0.1219
	Gaussian + score	0.0563	0.0746	0.8396	0.1219
	Cosine + score	0.0563	0.0746	0.8396	0.1219
90% labeled 10% unlabeled	Gaussian	0.0567	0.0723	0.8432	0.1196
	Gaussian + score	0.0567	0.0724	0.8429	0.1197
	Cosine + score	0.0567	0.0724	0.8429	0.1197

Table 3.6: Performance of FME.

Data size	**Output scaling**	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% labeled 50% unlabeled	Yes	0.0603	0.0784	0.8236	0.1326
	No	0.0912	0.1233	0.7742	0.2284
70% labeled 30% unlabeled	Yes	0.0551	0.0727	0.8435	0.1133
	No	0.0859	0.1142	0.7647	0.2159
90% labeled 10% unlabeled	Yes	0.0554	0.0704	0.8464	0.1119
	No	0.0832	0.1097	0.7957	0.2121

Table 3.7: Effect of scaling the output of KFME (using a Laplacian based on Gaussian feature similarity).

Data size	Laplacian	MAE	RMSE	PC	ϵ-error
50% labeled 50% unlabeled	Gaussian	0.0603	0.0784	0.8236	0.1326
	Gaussian + score	0.0603	0.0784	0.8243	0.1325
	Cosine + score	0.0603	0.0783	0.8242	0.1323
70% labeled 30% unlabeled	Gaussian	0.0551	0.0727	0.8435	0.1133
	Gaussian + score	0.0550	0.0729	0.8445	0.1140
	Cosine + score	0.0550	0.0728	0.8446	0.1138
90% labeled 10% unlabeled	Gaussian	0.0554	0.0704	0.8464	0.1119
	Gaussian + score	0.0561	0.0710	0.8454	0.1140
	Cosine + score	0.0560	0.0709	0.8455	0.1138

Table 3.8: Performance of KFME using Laplacians based on different similarities.

Delta	MAE	RMSE	PC	ϵ-error
0.005	0.0575	0.0740	0.8385	0.1241
0.03	0.0569	0.0717	0.8475	0.1173
0.05	0.0568	0.0716	0.8454	0.1163
0.08	0.0564	0.0713	0.8451	0.1148
0.10	0.0561	0.0710	0.8454	0.1140
0.13	0.0559	0.0709	0.8453	0.1133
0.17	0.0557	0.0707	0.8458	0.1126
0.20	0.0556	0.0706	0.8462	0.1122

Table 3.9: KFME with different values of δ in score similarity (90/10 training/test partition and Laplacian based on Gaussian feature similarity and score similarity).

MAE	0.0751
RMSE	0.0926
PC	0.7988
ϵ-error	0.1714

Table 3.10: Performance of KFME on unseen data.

Data size	Features	MAE	RMSE	PC	ϵ-error
50% train 50%test	Raw	0.0615	0.0792	0.8133	0.1346
	Zwb	0.0645	0.0826	0.7962	0.1435
	Kernel zwb	0.0601	0.0785	0.8153	0.1293
70% train 30%test	Raw	0.0563	0.0734	0.8368	0.1180
	Zwb	0.0564	0.0748	0.8377	0.1219
	Kernel zwb	0.0563	0.0748	0.8368	0.1217
90% train 10%test	Raw	0.0572	0.0721	0.8375	0.1169
	Zwb	0.0565	0.0726	0.8416	0.1202
	Kernel zwb	0.0680	0.0887	0.7305	0.1592

Table 3.11: Results of linear ϵ-SVR applied to raw features and those obtained from the flexible semi-supervised embedding.

Distance	Features	Total	Class 1	Class 2	Class 3
Euclidean	Raw	0.5600	0.0952	0.6993	0.1154
	Zwb	0.6900	0.2857	0.7974	0.3846
Cosine	Raw	0.6000	0.1429	0.7451	0.1154
	Zwb	0.7600	0.0000	0.9935	0.0377

Table 3.12: Effect of using 1-NN classifier to the features obtained with the graph-based embedding. The table depicts the correct classification rates.

Class	Interval in SCUT-FBP	Number of images
1	(0.2,0.4)	43
2	(0.45, 0.65)	306
3	(0.7, 1)	53

Table 3.13: Here there are the three discrete classes used in the experiment of table 3.12.

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% train 50%test	Euclidean Minkowski Cosine	0.0891 0.0976 0.0902	0.1176 0.1305 0.1200	0.5924 0.4450 0.6120	0.2229 0.2482 0.2228
70% train 30%test	Euclidean Minkowski Cosine	0.0876 0.0934 0.0885	0.1168 0.1257 0.1176	0.6052 0.4930 0.6375	0.2148 0.2331 0.2153
90% train 10%test	Euclidean Minkowski Cosine	0.0909 0.0939 0.0891	0.1175 0.1220 0.1139	0.5668 0.5136 0.6377	0.2310 0.2366 0.2208

Table 3.14: Results of 1-NN using different distances.

Data size	**MAE**	**RMSE**	**PC**	**ϵ-error**
50% train / 50% test	0.0814	0.1086	0.5969	0.1984
70% train / 30% test	0.0727	0.0962	0.6867	0.1708
90% train / 10% test	0.0645	0.0827	0.7772	0.1429

Table 3.15: Results of Ridge Regression.

Data size	Kernel	MAE	RMSE	PC	ϵ-error
50% train	Linear	0.0615	0.0792	0.8133	0.1346
50%test	Gaussian	0.0612	0.0789	0.8170	0.1342
70% train	Linear	0.0563	0.0734	0.8368	0.1180
30%test	Gaussian	0.0560	0.0729	0.8393	0.1170
90% train	Linear	0.0572	0.0721	0.8375	0.1169
10%test	Gaussian	0.0561	0.0711	0.8434	0.1151

Table 3.16: Results of using ϵ-SVR with a linear and a Gaussian kernel.

Algorithm	MAE	RMSE	PC	ϵ-error
1-NN	0.0891	0.1139	0.6377	0.2208
RR	0.0645	0.0827	0.7772	0.1429
SVR	0.0561	0.0711	0.8434	0.1151
LGC	0.0830	0.1076	0.7390	0.2090
FME	0.0567	0.0723	0.8432	0.1196
KFME	0.0561	0.0710	0.8454	0.1140
Zwb + SVR	0.0680	0.0887	0.7305	0.1592

Table 3.17: Summary of the performances of the algorithms on SCUT-FBP with a 90%/10% data configuration.

Data size	MAE	RMSE	PC
Eastern	0.1536	0.1866	0.0000
Western	0.1512	0.1831	0.0000
Both	0.1524	0.1849	0.0000

Table 3.18: Results of the baseline (M^{2}B dataset).

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**
Eastern	Gaussian	0.1502	0.1827	0.2051
	Gaussian + score	0.1502	0.1837	0.1996
	Cosine + score	0.1501	0.1837	0.2006
Western	Gaussian	0.1438	0.1742	0.3499
	Gaussian + score	0.1423	0.1741	0.3431
	Cosine + score	0.1423	0.1741	0.3433
Both	Gaussian	0.1484	0.1801	0.2290
	Gaussian + score	0.1484	0.1814	0.2162
	Cosine + score	0.1483	0.1813	0.2169

Table 3.19: Results of LGC using different Laplacians (M^2B dataset).

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**
Eastern	Gaussian	0.1352	0.1668	0.4482
	Gaussian + score	0.1352	0.1668	0.4482
	Cosine + score	0.1352	0.1668	0.4482
Western	Gaussian	0.1141	0.1422	0.6338
	Gaussian + score	0.1141	0.1422	0.6338
	Cosine + score	0.1141	0.1422	0.6338
Both	Gaussian	0.1346	0.1665	0.4358
	Gaussian + score	0.1346	0.1665	0.4358
	Cosine + score	0.1346	0.1665	0.4358

Table 3.20: Performance of FME (M^2B dataset).

Data size	**Laplacian**	**MAE**	**RMSE**	**PC**
Eastern	Gaussian	0.1358	0.1671	0.4455
	Gaussian + score	0.1357	0.1670	0.4466
	Cosine + score	0.1357	0.1669	0.4469
Western	Gaussian	0.1132	0.1424	0.6322
	Gaussian + score	0.1134	0.1424	0.6319
	Cosine + score	0.1133	0.1419	0.6362
Both	Gaussian	0.1303	0.1624	0.4805
	Gaussian + score	0.1302	0.1623	0.4806
	Cosine + score	0.1302	0.1624	0.4800

Table 3.21: Performance of KFME using Laplacians based on different similarities (M^{2}B dataset, 50%/50% training/test partition).

Delta	**MAE**	**RMSE**	**PC**
0.005	0.1358	0.1668	0.4503
0.03	0.1358	0.1671	0.4457
0.05	0.1358	0.1670	0.4461
0.08	0.1357	0.1670	0.4466
0.1	0.1357	0.1670	0.4466
0.13	0.1357	0.1670	0.4466
0.17	0.1357	0.1670	0.4461
0.2	0.1357	0.1670	0.4462

Table 3.22: KFME with different values of δ in score similarity (M^{2}B dataset, Eastern images, 50/50 training/test partition).

Distance	**Features**	**Total**	**Class 1**	**Class 2**	**Class 3**
Euclidean	Raw	0.6150	0.0870	0.7580	0.1000
	Zwb	0.6450	0.1739	0.7834	0.1000
Cosine	Raw	0.6150	0.0870	0.7580	0.1000
	Zwb	0.7450	0.0870	0.9299	0.0500

Table 3.23: Effect of using 1-NN classifier to the features obtained with the graph-based embedding. The table depicts the correct classification rates (M^2B dataset).

Data size	**Features**	**MAE**	**RMSE**	**PC**
Eastern	Raw	0.1356	0.1679	0.4397
	Zwb	0.1350	0.1672	0.4460
	Kernel zwb	0.1341	0.1670	0.4577
Western	Raw	0.1144	0.1435	0.6239
	Zwb	0.1138	0.1421	0.6349
	Kernel zwb	0.1125	0.1413	0.6369

Table 3.24: Results of linear ϵ-SVR applied to raw features and those obtained from the flexible semi-supervised embedding v.

Class	**Interval in M^2B**	**Number of images**
1	[0.1, 0.3)	46
2	(0.44, 0.72)	315
3	(0.85, 1]	41

Table 3.25: The three discrete classes used in the experiment of table 3.12.

Chapter 4

Conclusion

4.1 Discussion of the results

We have seen how to apply different graph-based semi-supervised methods to the task of automatic beauty assessment, which can be particularly interesting if one considers the lack of studies based on semi-supervised learning in this area. It has been proved that three label propagation methods, LGC, FME and KFME, which were firstly designed to perform classification tasks, can also be adapted, with different degrees of success, to work on regression. LGC produces the worst results, not only because it is the simplest one of the three, but also due to the number of problems which arise when adapting the idea to regression (mentioned in section 2.1.2). Contrariwise, KFME has been the best method. It slightly improves the best PC of the original study on SCUT-FBP [22], which was 0.8187, while ours is 0.8464.

In figure 3.6, one can see a graphical comparison of the algorithms in the case of a 70%/30% training/test data partition on SCUT-FBP dataset, where KFME, which corresponds to the steepest curve, outperforms the rest of the algorithms. Regarding the rest of the algorithms, FME seems to work similarly to ϵ-insensitive SVR, which works slightly better than Ridge Regression. LGC is similar to 1-NN, although it is a bit better with a 90%/10% configuration.

It is also interesting to remember that a crucial step in this process has been scaling the output of KFME (recall table 3.7). This need of scaling seems to come from the mathematical formulation of the problem. If one compares the equations describing FME and KFME (equations 2.4 and 2.5 in sections 2.1.3 and 2.1.4), it can be noticed that, while in the regression term of the former there is a constant b, there is no such variable in KFME, so this could be the reason of producing biased and incorrectly scaled results.

The last semi-supervised method, the graph-based embedding, has not provided so satisfactory results in SCUT-FBP, although in M^2B the results seem a bit more promising. As suggested by tables 3.11 and 3.12, the reason could be the fact that we are working on regression rather than classification. Indeed, if we take into account that the embedding clusterizes the data into classes (in our case, 5 classes), it seems very reasonable to assume that this will enhance the performance of a classification algorithm, because classes are more separated. However, if we consider the

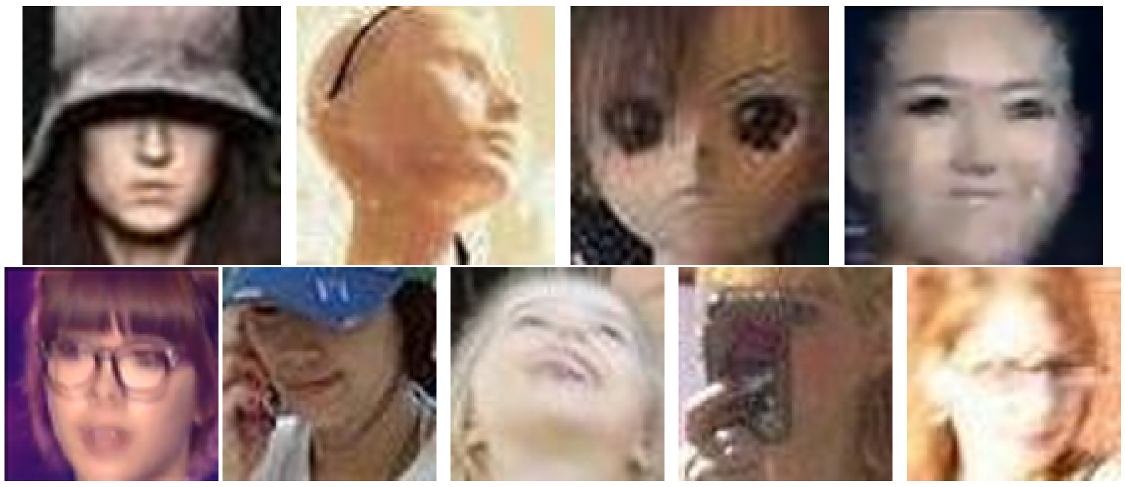


Figure 4.1: Examples of images difficult to rank in M^{2}B dataset.

regression task, this embedding seems to increase just the probability of hitting the class, but not the probability of predicting the specific label with all its decimals.

If we have a look at the results on M^{2}B dataset, the first thing we will notice is that they are substantially worse than in SCUT-FBP. A very plausible explanation is the quality of the data. In my modest opinion, figure 4.1 is sufficiently illustrative of this fact. Although most of the images are acceptable, images similar to those shown in the figure can negatively affect the algorithms' performances.

Table 4.1 shows the comparison between the best MAEs of the original study on M^{2}B dataset and ours. Our results are multiplied by 10, because we used the normalized labels (dividing the original labels by 10). Our worst algorithm (LGC) outperforms their best algorithm, even if both studies have been carried out using the same training/test proportion. We were using 10 splits with a 50%/50% labeled/unlabeled configuration, while they were using 2-fold cross validation.

4.2 Future work

Considering that the study of different similarity graphs has not led us to conclusive results - because sometimes graphs considering only feature similarity work better than those also taking into account score similarity and vice versa - it could be a good idea to try more sophisticated similarity graphs. For example, instead of considering the binary score similarity described in section 2.1.6,

$$s_{ij}^{score} = \begin{cases} 1 & \text{if } |y_i - y_j| < \delta \\ 0 & \text{otherwise,} \end{cases}$$

where y_i and y_j are the labels of instances i and j or, in other words, the means of the ratings of all raters of images i and j respectively, a more elaborated similarity could be used. For example, the standard deviations of the ratings of each of the

OUR STUDY		
	Eastern	**Western**
LGC	1.501	1.423
FME	1.352	1.141
KFME	1.357	1.132
Zwb + SVR	1.341	1.125

ORIGINAL STUDY		
	Eastern	**Western**
1-NN	2.11	1.92
Ridge Regression	1.95	1.87
Neural Network	1.82	1.76
F-A-T	1.80	1.69
DFAT	1.77	1.66

Table 4.1: Comparison between our study on M^{2}B dataset and the original one [13].

samples, σ_i and σ_j could be used, leading to a formula such as

$$s_{ij}^{score} = e^{-\frac{|y_i - y_j|}{\sigma_i \sigma_j}}.$$

So, the a hybrid similarity considering both feature and score information could be the product of both similarities:

$$s_{ij} = s_{ij}^{feature} \cdot e^{-\frac{|y_i - y_j|}{\sigma_i \sigma_j}},$$

where $s_{ij}^{feature}$ can be either the cosine similarity or the Gaussian feature similarity described in section 2.1.6.

Future work may also envision revisiting the design of the criterion of the graph-based non-linear embedding (described in section 2.1.5). Specifically, we would like to redefine the matrix $\tilde{\mathbf{M}}_l$, so that the margin related term could take into account the distance in the score space.

Bibliography

[1] Parham Aarabi et al. “The automatic measurement of facial beauty”. In: *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*. Vol. 4. IEEE. 2001, pp. 2644–2647.

[2] Jinbo Bi and Kristin P Bennett. “Regression error characteristic curves”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 43–50.

[3] Fadi Dornaika and Youssof El Traboulsi. “Learning flexible graph-based semi-supervised embedding”. In: *IEEE transactions on cybernetics* 46.1 (2016), pp. 206–218.

[4] Yael Eisenthal, Gideon Dror, and Eytan Ruppin. “Facial attractiveness: Beauty and the machine”. In: *Neural Computation* 18.1 (2006), pp. 119–142.

[5] Youssof El Traboulsi, Fadi Dornaika, and Ammar Assoum. “Kernel flexible manifold embedding for pattern classification”. In: *Neurocomputing* 167 (2015), pp. 517–527.

[6] Junying Gan et al. “Deep self-taught learning for facial beauty prediction”. In: *Neurocomputing* 144 (2014), pp. 295–303.

[7] Douglas Gray et al. “Predicting facial beauty without landmarks”. In: *Computer Vision–ECCV 2010* (2010), pp. 434–447.

[8] Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[9] Amit Kagian et al. “A humanlike predictor of facial attractiveness”. In: *Advances in Neural Information Processing Systems*. 2007, pp. 649–656.

[10] Amit Kagian et al. “A machine learning predictor of facial attractiveness revealing human-like psychophysical biases”. In: *Vision research* 48.2 (2008), pp. 235–243.

[11] Shu Liu et al. “Advances in computational facial attractiveness methods”. In: *Multimedia Tools and Applications* 75.23 (2016), pp. 16633–16663.

[12] Yadong Mu. “Computational facial attractiveness prediction by aesthetics-aware features”. In: *Neurocomputing* 99 (2013), pp. 59–64.

[13] Tam V Nguyen et al. “Towards decrypting attractiveness via multi-modality cues”. In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.4 (2013), p. 28.

[14] Feiping Nie et al. “Flexible manifold embedding: A framework for semi-supervised and unsupervised dimension reduction”. In: *IEEE Transactions on Image Processing* 19.7 (2010), pp. 1921–1932.

[15] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. "Deep Face Recognition." In: *BMVC.* Vol. 1. 3. 2015, p. 6.

[16] Kendra Schmid, David Marx, and Ashok Samal. "Computation of a face attractiveness index based on neoclassical canons, symmetry, and golden ratios". In: *Pattern Recognition* 41.8 (2008), pp. 2710–2717.

[17] Ali Sharif Razavian et al. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops.* 2014, pp. 806–813.

[18] Alex J Smola and Bernhard Schölkopf. "A tutorial on support vector regression". In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[19] Alex J Smola and Bernhard Schölkopf. *Learning with kernels.* GMD-Forschungszentrum Informationstechnik, 1998.

[20] V Vapnik. *The Nature of Statistical Learning Theory.* 1995.

[21] Shuyang Wang, Ming Shao, and Yun Fu. "Attractive or not?: Beauty prediction with attractiveness-aware encoders and robust late fusion". In: *Proceedings of the 22nd ACM international conference on Multimedia.* ACM. 2014, pp. 805–808.

[22] Duorui Xie et al. "SCUT-FBP: A benchmark dataset for facial beauty perception". In: *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on.* IEEE. 2015, pp. 1821–1826.

[23] Jie Xu et al. "A new humanlike facial attractiveness predictor with cascaded fine-tuning deep learning model". In: *arXiv preprint arXiv:1511.02465* (2015).

[24] Denny Zhou et al. "Learning with local and global consistency". In: *Advances in neural information processing systems.* 2004, pp. 321–328.