Reducing Dimensionality for Face Recognition
Miniproject for Machine Learning and Data Mining (COMP61011)

Nikolaos Nikolaou (8101425) NIKOLAON@CS.MAN.AC.UK
CDT Computer Science, University of Manchester

Abstract

In this project we explore various aspects involved in a face recognition task, including preprocessing the data, feature extraction, dimensionality reduction and feature selection, the evaluation and comparison of the performance of various classifiers, as well as basic methods for selecting appropriate hyperparameter values and a demonstration of how the performance of weak learners can be improved via meta-algorithms such as AdaBoost. We will compare the performance of k-Nearest Neighbour, Naive Bayes, Decision Trees and AdaBoosted Decision Trees using different feature sets including the full set of pixel intensities, Eigenfaces, Fisherfaces and subsets of pixels selected by two different feature selection criteria: Mutual Information and Joint Mutual Information. Special emphasis will be given to reducing the dimensionality of the input space.

1. The Task and the Dataset

In this project we will examine a classification problem in the domain of image –and in particular face – recognition. We will be working with the *Extended Yale Face Database B* [1] and more precisely the *Cropped Yale Dataset* version [2]. The latter contains 2424 images of 38 human subjects, each captured under 64 illumination conditions (some images were corrupted during the image acquisition, thus not all 2432 are included) which have been manually aligned, cropped, and then re-sized to 168×192 pixels.

Initially we have a feature matrix $\boldsymbol{X} \in \mathbb{R}^{32256\times 2424}$, each column being a different feature vector (each images' columns were concatenated to form one big vector). Conversely, we also have a corresponding label vector $\boldsymbol{y} \in \mathbb{R}^{2424}$ with values $y_i \in [1, 38]$, where the number refers to the subject to whom the image belongs. For this project, we resized the images by a factor of $r = 0.4$ which resulted in images of 77×68 pixels. This meant that the initial feature matrix was $\boldsymbol{X} \in \mathbb{R}^{5236\times 2424}$. Working with these rescaled images was much more cheap computationally. Rescaling, of course, leads to loss of information.

On each run, we randomly split the dataset into two parts: the training set ($N_train = 2000$ instances) and the test set ($N_test = 424$ instances). We train our classifiers on the training set and optimize *hyperparameters* using a *5-fold cross-validation (CV)* scheme. We then evaluate them on test (hold out) set. In order to obtain the appropriate statistics, we perform this entire procedure 20 times in each case. The task is a *multiclass classification* problem. Given a new feature vector $\boldsymbol{x} \in \mathbb{R}^{5236}$ we would like our classifier to find to which of the 38 subjects it belongs. We will denote the classes by $C_i, i = 1, 2, ..., 38$.

Preprocessing: Aside from the rescaling of the images, which is done for reasons of memory and time complexity, another important step is *standardizing* the data, i.e. subtracting from each feature its mean and dividing it by its standard deviation. This normalization is performed on every instance and it is crucial as the difference in scaling of the features can affect many of the techniques we discuss.

Another form of preprocessing we applied to the dataset is *discretizing* (*quantizing*) the attribute values. This was necessary to do in order to calculate the *entropies* involved in the computation of *mutual information* and *joint mutual information*, as the toolboxes used worked on discrete features. In section 2 we briefly cover the details of this process, some additional benefits of using quantized features and the method of choosing the number of bins.

2. Features Examined

Pixel intensities: This is the most obvious choice of features. However, using pixel intensities as features also has a number of drawbacks. First of all, the dimensionality of the feature vector (32256 features before rescaling, 5236 in the shrunk version we used) causes time and memory complexity issues. These

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

might be inconsequential in the context of our project, however in real life applications using pixel intensities as pixels usually renders the computational costs involved prohibitive. Furthermore, there are other potential disadvantages. For instance, *Naive Bayes (NB)* assumes conditional independence among the features given the class label. This assumption doesn't hold in our case as pixels within an image are certainly not independent (not only neighbouring ones, faces also exhibit bilateral symmetry). In Figure 2[**Bottom**] we see a scatterplot of the values assumed by two adjacent pixels indicating significant correlation between them.

Pixel intensities (discretized): By using discretized pixel intensities as features we might be able to filter out some of the irrelevant information carried in the continuous intensity values and allow for better generalization. The problems related to dimensionality are not solved, however, as the feature vector still has the same dimensions. We used bins of equal width and did a line search for the optimal bin number $L \in \{50, 40, 30, 20, 10\}$ evaluating the choices using *5-fold CV* on the 2000 training instances. We repeated the procedure 10 times for different permutations of the training data. In Figure 5[**Bottom**] we can see the resulting average *CV* accuracy attained for each choice of L. We ended up using 20 bins. Our approach is somewhat naive. We could have achieved a better discretization using binsizes of variable length, e.g. based on the histogram of the data, in such a way as to have each bin contain a set of values that occurs with equal frequency.

Another benefit they offer comes into play when using the *NB* classifier. When working with continuous features we use the "Gaussian version" of the *NB* classifier. As its name suggests, it assumes that the feature probability distributions (for each class) are Gaussian with parameters the corresponding means and variances computed from the training set. If we know that the underlying distributions are normal then this assumption holds. When having a small number of features or continuous features and no prior knowledge of the form of the underlying distributions, assuming that they are Gaussian is fairly reasonable[1]. However, when having a fairly large amount of data (as in our case) we can avoid this unnecessary (and likely invalid, as implied by plotting the class conditional histogram of an arbitrary pixel for an arbitrary class as shown in Figure 2[**Top**]) assumption and compute the actual distribution straight from the data by calculating the appropriate relative frequencies of occurrences [2]. In order for this procedure to make sense, the data have to be discretized first. So, with our *discretized* data, we use the version "non-Gaussian assuming version" of *NB*.

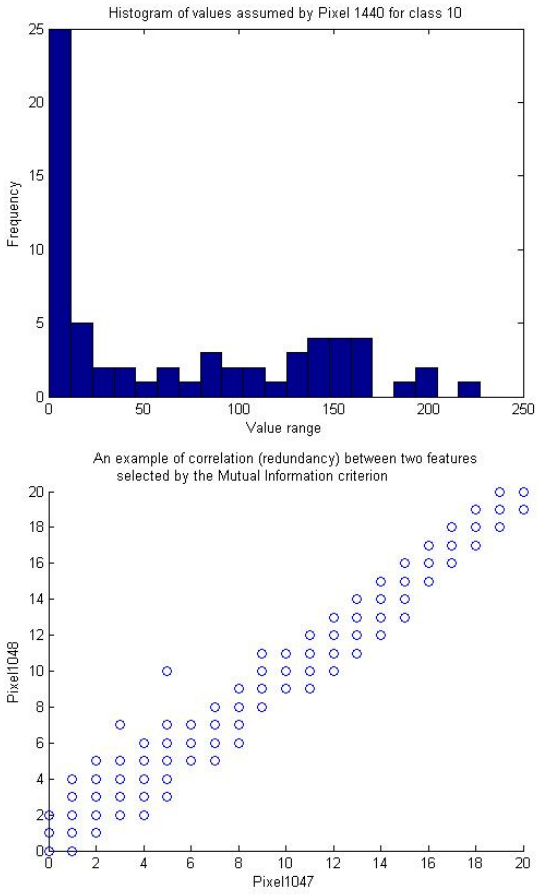

Figure 1. [**Top**] The class-conditional histogram of an arbitrary pixel in instances of an arbitrary class. The assumption that the values follow a Gaussian distribution seems to be invalid. [**Bottom**] A scatterplot of two adjacent pixels selected by *MI* as *relevant*. We notice significant linear correlation between them. This indicates that the conditional independence assumption of the NB classifier does not hold. Also, it shows us that MI does not factor in the redundancies among features.

Finally, let us try to give an explanation of how the number of bins can impact our *NB* model's generalization properties as we implied above. Suppose we use L quantization levels $\{b_1, b_2, ..., b_L\}$ on our 38-class problem. We need to compute parameters $P(C_j), \forall j \in \{1, 2, ..., 38\}$ (class priors) as well as $P(b_i|C_j), \forall i \in \{1, 2, ..., L\}, \forall j \in \{1, 2, ..., 38\}$ (to model the class conditional probabilities). We can think of L as a *hyperparameter* of our model that controls how many parameters we need to describe our model. Larger values of L mean more $P(b_i|C_j)$ terms, thus a more *complex*

[1]Some justification is offered by the Central Limit Theorem, also, working with Gaussians is convenient.

[2]In both cases we perform Maximum Likelihood Estimation to compute the parameters of the distribution, the difference is that in the second case we do not assume it is Gaussian.

model, while smaller values of L mean less $P(b_i|C_j)$ terms, thus a less *complex* model. As we know, the *complexity* of a model is associated with its *generalization* capability[3]. In this case the number of bin sizes L controls the *complexity*. We use the average CV error computed in the *hyperparameter selection* phase as an estimate of the *generalization error* and choose the value of L, that minimizes this [4].

3. Reducing the Dimensionality

One of the greatest challenges in image recognition applications is the high dimensionality of the dataset. It is too expensive computationally to use the entire vector of pixel intensities as features. Furthermore, there are a number of non-computationally related implications that arise in situations of high dimensional data which affect many machine learning methods, collectively referred to as the *Curse of Dimensionality*[5].

As a result, we need to find ways to reduce the dimensionality of the dataset. We will explore solutions that fall into two different groups of approaches here: *feature extraction* and *feature selection*. *Feature extraction*, consists of combining (e.g. linearly) the initial D features $\{x_1, x_2, ..., x_D\}$ to form $K < D$ new ones $x' = \{x'_1, x'_2, ..., x'_K\}$ through appropriate transformations $x'_i = \Phi_i(x_1, x_2, ..., x_D), \forall i = 1, ..., K$. *Feature selection* consists of selecting an appropriate subset of the initial features, i.e. $x' \subset x$, again reducing the dimensionality of the feature space from D to $K < D$. Two common approaches to feature selection are *wrapper techniques* and *filter-based methods*. *Wrappers* search the feature space using a *search algorithm* (e.g. *exhaustive search*, *greedy search methods*, *simulated annealing*, *genetic algorithms*, etc.) by training and evaluating a model on subsets of features. With this many features, this approach would be too demanding computationally. Another approach is the use of *filters* to 'filter out' features that are deemed less useful by some criterion.

[3]Too simple models fail to capture the underlying patterns and end up *underfitting* the training data. Too *complex* models fail to ignore the *noise* within the training set. In this case we *overfit* our model to the training data, again failing to *generalize*. We have to find the appropriate "sweet-spot" for the *complexity* of our model so as to minimize the (expected estimated) generalization error.

[4]Equivalently maximizes the average CV accuracy.

[5]When the dimensionality increases, the volume of the feature space increases exponentially, thus the available data becomes sparse. Machine learning techniques usually rely on detecting areas of the feature space where instances form groups with similar properties. In high dimensional feature spaces, however (unless having vast amounts of training examples) the instances appear sparse and dissimilar to one another, thus hindering the extraction of useful patterns.

As *feature extraction* methods, we will examine two *linear* transformations of the initial dataset: *Principal Components Analysis (PCA)* and *Linear Discriminant Analysis (LDA)*. As *feature selection* methods, we will examine two information-theoretical criteria for filtering the data: *Mutual Information (MI)* and *Joint Mutual Information (JMI)*.

Principal Components Analysis: By performing *PCA* we aim to decrease the dimensionality of the data by projecting them on a space defined by $K < D$ *Principal Components (PCs)*. The main idea is to find the directions that account for the greatest variation among the data. Thus, the first *component* corresponds to the direction of the greatest variability in the data, the second *component* is chosen as *perpendicular* to the first and captures the most of what remains of the variability in the data, and so on until we reach the D-th *component*. We then choose the first $K < D$ PCs and project the data on these.

To perform *PCA* on the data $\boldsymbol{X} \in \mathbb{R}^{D \times N}$, where N is the number of instances, we first compute the covariance matrix covariance matrix $\boldsymbol{Cov(X)} \in \mathbb{R}^{D \times D}$ of $\boldsymbol{X}$ by $Cov(\boldsymbol{X}) = \big[Cov(x_i, x_j)\big] \, \forall i, j = \big[E\left[(x_i - \mu_i)(x_j - \mu_j)\right]\big] \, \forall i, j$, where x_k is the k-th feature, $\mu_k = E(x_k)$ is its mean and $i, j \in \{1, 2, ..., D\}$. Then we need to compute the eigenvalues $\lambda_k, k = 1, 2, ..., D$ and corresponding eigenvectors $\boldsymbol{u_k}, k = 1, 2, ..., D$ of $\boldsymbol{Cov(X)}$. We sort the eigenvectors according to their corresponding eigenvalues in descending order. This allows us to represent our original data matrix as $\boldsymbol{X} = \boldsymbol{U\Lambda V^T}$, where $\boldsymbol{U} \in \mathbb{R}^{D \times D}$ is a matrix with the eigenvectors $\boldsymbol{u_k}, k = 1, 2, ..., D$ of the covariance matrix as its columns and $\boldsymbol{\Lambda} \in \mathbb{R}^{D \times D}$ is a diagonal matrix with the corresponding eigenvalues $\lambda_k, k = 1, 2, ..., D$ as its diagonal elements (λ_1 is the largest eigenvalue, λ_2 the second largest, etc.).

We then choose the K first columns of $\boldsymbol{U}$ and discard the rest, thus resulting in a new basis matrix $\boldsymbol{E} \in \mathbb{R}^{D \times K}$. In our implementation we chose the K *principal components* that accounted for (at least) 95% of the variance within the data. We did this by selecting the smallest K such that $\frac{\sum_{k=1}^{K} \lambda_k}{\sum_{k=1}^{D} \lambda_k} \leq 0.95$. We project the data $\boldsymbol{X}$ onto the new basis matrix $\boldsymbol{E}$ and end up with a new feature representation $\boldsymbol{X'} = \boldsymbol{E^T X}$, where $\boldsymbol{X'} \in \mathbb{R}^{K \times N}$.

By performing *PCA* on our dataset and projecting the data onto the new basis formed by the eigenvectors of the covariance matrix, we have essentially applied the *Eigenfaces* approach to face recognition. The eigenvectors are, in this context called "*Eigenfaces*" and each face is projected on the *eigenface space*, therefore it is reconstructed as a *linear combination* of these *Eigenfaces*. Thus, the *Eigenfaces* can be perceived as a set of "face templates". Any face in the dataset can

be considered to be equal to the "mean face", plus a weighted sum of these templates. Since they are the eigenvectors which correspond to the largest eigenvalues of the covariance matrix, the *Eigenfaces* capture the dimensions characterized by the greatest variability in our data. In Figure 3 we see the percentage of variance explained as we add more *PC*s in a single test run. Here we selected the first $K = 53$ *PC*s.

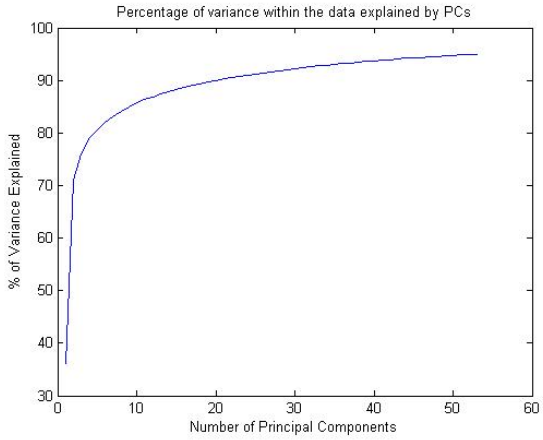

Figure 2. The percentage of variance within the dataset explained as we add more PCs. Notice that the first PC accounts for more than 35%. Using just 10 accounts for more than 80% and for 95% we only need 53 of the total 5236 components.

PCA is an *unsupervised* technique. At no step do we make use of the class labels of the training set which is information that is known to us. Therefore, *PCA* only cares about preserving the variability of the original dataset in a smaller-dimensional space. It is not guaranteed to offer better class discrimination. In fact, it can even choose directions that make it harder to separate classes. For example, in retrospect, we should have refrained from using the first 3 *PC*s as features as done in [4]. These *PC*s correspond to variations due to lightning conditions and not to actual *between-class variations*[6]. Our results would have probably been better had we done so. The rest of the *components* capture actual variations of faces including position and shape of the eyes, width of mouth, size and shape of nose and eyebrows, etc.

We should clarify that the *PC*s were calculated from the training data and then the entire dataset (both training and test sets) was projected on the *eigenspace*[7]. As a final note, it was necessary to *standardize* the data before performing *PCA*, as the scaling of the features affects which directions are chosen as *PC*s (large values would dominate the covariance matrix).

Linear Discriminant Analysis: *Fisher's Linear Discriminant Analysis* is another way to obtain a lower-dimensional representation of the original data [5]. *LDA* contrary to *PCA* is a *supervised* technique, i.e. it makes use of the class label information of the training data. However *LDA* also has its own drawbacks. First of all it assumes that the class conditional feature distributions are Gaussian. We saw that this assumption here is dubious. Finally, *LDA* will fail if the class-discriminatory information is not in the mean itself, but in the variance of the data.

Assuming the data in each class are normally distributed, the aim of *LDA* is specifically to find the lower dimensional (linear) manifold that best discriminates the classes. Our goal is to find the directions that minimize the *within-class variance* $S_w = \sum_{k=1}^{38} \sum_{i=1}^{n_k} (x_{i,k} - \mu_k)((x_{i,k} - \mu_k)^T$ and maximize the *between-class variance* $Sb = \sum_{k=1}^{38} (\mu_k - \mu)((\mu_k - \mu)^T$, where $x_{i,k}$ is the i-th sample of class k, n_k is the number of samples in class k, μ_k is the mean of the feature for instances belonging to class k, μ is the overall mean and $i = 1, 2, ..., N_{train}$.

We need to compute matrix U, whose columns will form the basis vectors u_i of the new feature space, such that S_b is maximized and S_w is minimized. We compute it by solving the *generalized eigenvalue decomposition* problem $S_b U = S_w V \Lambda$, where Λ is the diagonal matrix of the corresponding eigenvalues.

Then we only need retain the K eigenvectors corresponding to non-zero eigenvalues and discard the rest. In this context the eigenvectors selected are called *Fisherfaces*. Finally, we project the data $\boldsymbol{X}$ onto the *fisherspace* $\boldsymbol{F} \in \mathbb{R}^{D \times K}$ and obtain a new feature representation $\boldsymbol{X'} = \boldsymbol{F^T X}$, where $\boldsymbol{X'} \in \mathbb{R}^{K \times N}$.

In our implementation we first projected the original faces on the *eigenspace* and then performed *LDA*. This is a usual tactic in image related tasks as in these problems the number of features (initially pixels) is much greater than the number of instances, thus matrix S_w is *singular* (has no *inverse*). By performing *LDA* on the *eigenspace* we avoid this problem as the dimensionality of the *eigenspace* is low.

Feature Selection by Mutual Information: The *Mutual Information* between the j-th feature $\boldsymbol{x_j} \in \mathbb{R}^N$ and the class label $\boldsymbol{y} \in \mathbb{R}^N$ can be expressed as $MI(\boldsymbol{x_j}) = I(\boldsymbol{x_j}; \boldsymbol{y}) = H(\boldsymbol{x_j}) - H(\boldsymbol{x_j}|\boldsymbol{y}) = H(\boldsymbol{y}) - H(\boldsymbol{y}|\boldsymbol{x_j})$. *Entropy* is a measure of uncertainty, so the *marginal entropy* $H(\boldsymbol{x_j})$ can be regarded as the un-

[6] We did not include images of Eigenfaces (or Fisherfaces) due to lack of space. We showed examples during the lab demonstration.

[7] We regarded the test instances as "unseen", until it is time for their own classification. Similarly, no information from the test data was used for *standardizing* and *discretizing* the data, performing *LDA*, and *feature selection*.

certainty of the random variable $\boldsymbol{x_j}$. The *conditional entropy* $H(\boldsymbol{x_j}|\boldsymbol{y})$ is a measure of the uncertainty in $\boldsymbol{x_j}$ given the class label $\boldsymbol{y}$. Thus, $I(\boldsymbol{x_j};\boldsymbol{y})$ measures the amount of uncertainty in $\boldsymbol{x_j}$ which is removed by knowing $\boldsymbol{y}$. Of course as we saw in the definition $MI(\boldsymbol{x_j})$ it also measures the amount of uncertainty in $\boldsymbol{y}$ given $\boldsymbol{x_j}$ as well (hence *"mutual" information*).

Based on this, we can use $I(\boldsymbol{x_j};\boldsymbol{y})\forall j \in \{1,2,...,D\}$ to calculate how explanatory feature $\boldsymbol{x_j}$ is of the class label. We can then select just a subset of the initial features, based on how well they differentiate individuals. However, doing so would only give us the *individually most relevant features*. We should keep in mind that by examining only one feature at a time we may select *redundant* features (e.g. highly correlated features, one of which would have sufficed) or fail to select combinations of features that only when combined allow us for better class discrimination.

In our implementation we discard the 50% lower ranking features (pixels) and use only the $D/2$ most *relevant* features as ranked by the *MI* criterion. In Figure 3**[Top]**, we visualize the position of the selected pixels. The position of the pixels makes perfect sense, as it corresponds to areas of the face that indeed differentiate individuals: eyes, brows, nose(and in particular the distance between nostrils), mouth, facial hair, face outline. Conversely, areas that are more or less the same for all persons, like the cheeks, are ignored and their corresponding pixels are discarded.

Feature Selection by Joint Mutual Information: When performing *feature selection* we do not only care about maximizing the *relevance* of the selected subset of features, but also about minimizing the *redundancy* within it. *Joint Mutual Information* [6] also takes into account the *redundancy* of using pairs of pixels as features, in conjunction with the *relevance* of each pair of features for predicting the class. *JMI* is defined as $JMI(\boldsymbol{x_j}) = \sum_{\boldsymbol{x_j}\in S} I(\boldsymbol{x_j x_k};\boldsymbol{y})$, where by S we denote the set of all features.

We computed $JMI(\boldsymbol{x_j})\forall j \in \{1,2,...,D\}$ using the *feast()* toolbox provided as part of the *MLOtools*. Again, combinations of three or more features are not examined, however *pairwise redundancies* have been avoided and pairs of pixels that when combined offer better class discrimination have been examined and possibly included. We used the top 50% ranking features and discarded the rest. As we can see, again the selected features are situated around the eye, nose, mouth and face outline areas. This time, however the *redundancy* due to the face's bilateral symmetry is taken into account and thus the features are selected from the one side of the face, with only pixels from very important areas, such as the eyes and mouth derived from both "symmetrical" sides[8].

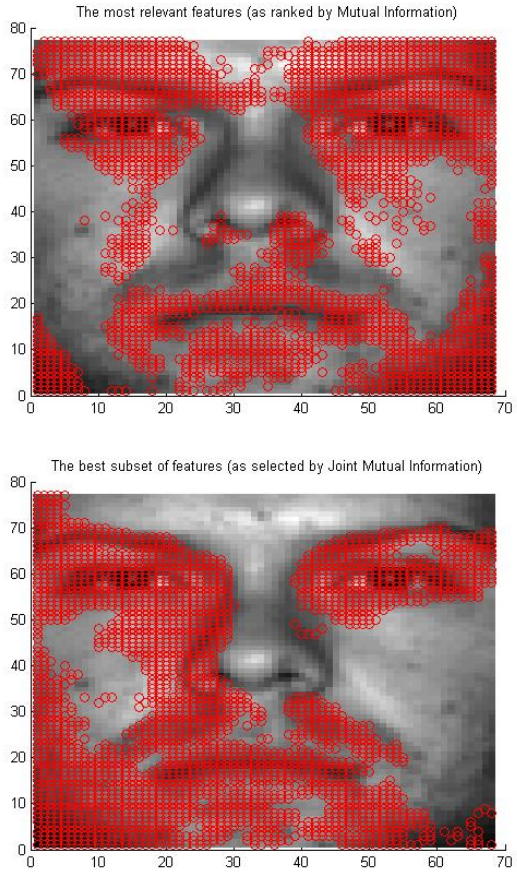

Figure 3. **[Top]** 50% most relevant features selected by MI. [Bottom] Top 50% features as ranked by JMI. Selected pixels are marked with red circles.

4. Classifiers

Naive Bayes (NB): The first classifier we study is the *Naive Bayes* Classifier. The goal here is to compute the *posterior probabilities* $P(C_k|\boldsymbol{x_i})\forall k = \{1,2,...,38\}$, i.e. the probability that test a example belongs to class C_k given that its feature values vector is $\boldsymbol{x_i} \in \mathbb{R}^D$. To compute this, first we use *Bayes' Rule* (hence the "Bayes" part of its name) and express this as the product of the *prior probability* $P(C_k)$ of class C_k and the *likelihood* $P(\boldsymbol{x_i}|C_k)$ (i.e. the prob-

[8]Illumination conditions could force some faces to 'break their symmetry' in these particular areas. Light coming towards their face from one side, might have caused **some** of the subjects to "squint" their eye on this side thus also slightly bending the lips. Hence pixels from both eyes and sides of the mouth were selected as they can help differentiate among individuals since not all subjects exhibited this reaction

ability of having an instance with feature values $\boldsymbol{x_i}$ given it belongs to class C_k), divided by a *normalization factor* $P(\boldsymbol{x_i}) = \sum_{k=1}^{38} P(\boldsymbol{x_i}|C_k)$. The *normalizer* is common for all classes to ensure that the posterior is a probability distribution. We then introduce the assumption that the features $x_{i,j} \forall j \in \{1, 2, ..., D\}$ are *conditionally independent from one another given the class label*. This is the origin of the "Naive" part of the classifier's name, as this assumption often doesn't hold (yet had we not made it the computational complexity of computing $P(C_k|\boldsymbol{x_i})$ would have been prohibitive). So we have $P(C_k|\boldsymbol{x_i}) = \frac{P(C_k)P(\boldsymbol{x_i}|C_k)}{\sum_{k=1}^{38} P(\boldsymbol{x_i}|C_k)} = \frac{P(C_k)\prod_{j=1}^{D} P(x_{i,j}|C_k)}{\sum_{k=1}^{38}\prod_{j=1}^{D} P(x_{i,j}|C_k)}$. We simply assign example $\boldsymbol{x_i}$ to the class C^* such that $C^* = \arg\max_C P(C|\boldsymbol{x_i})$ or equivalently, $C^* = \arg\max_C P(C)\prod_{j=1}^{D} P(x_{i,j}|C)$. As, since the denominator is the same for all classes, we can ignore it when maximizing.

We already discussed in Section 1 how the training (i.e the computation of the optimal values of the parameters of the model for it to best fit the data)is performed. We make use of the *gnbayes()* and *nbayes()* implementations of *MLOtools*.

k Nearest Neighbour: Next, we examined two variants of the *k-Nearest Neighbour* algorithm. The first one is the *classic kNN* version, where to for each test example $\boldsymbol{x_i} \in \mathbb{R}^D$, we compute its distance from all N_{train} training examples, then we find the class labels of its k *nearest neighbours* and classify $\boldsymbol{x_i}$ to the class the majority of these neighbours belong to. The second one, instead of choosing the predicted label of an incoming instance through a *majority vote* among its k *nearest neighbours*, uses a *weighted vote* instead. Each neighbour has a weight equal to its inverse distance from the incoming test example. Both versions were implemented from scratch and both use the *euclidean distance*. In both algorithms k was selected by CV as described in Section 1 and ties are resolved by calling the corresponding version of $1NN$.

4.1. Decision Trees (DT)

The *Decision Trees* were implemented using the built-in MATLAB *classregtree()* function. On each level the feature upon which to split is selected based on its *Gini's diversity index*. No restriction was imposed regarding the *maximal depth* of the trees and default *pruning* was applied to prevent the trees from *overfitting*. We also used DTs as *base classifiers* for *AdaBoost* (see section 6) achieving a significant improvement in classification accuracy.

5. Hyperparameter Selection

We performed a line search for the optimal $k \in \{1, 3, 5, 7, 9, 11, 13, 15\}$ for the two *kNN* variants examined. We select the one which leads to the best average classification accuracy as computed by performing *5-fold CV* on the training data. We repeated the procedure 20 times for different permutations of the training data. The best results were obtained for $k = 7$. We use the exact same procedure in order to select the optimal bin number for the *discretization* of the data. An illustration of the results is shown in Figure 5.

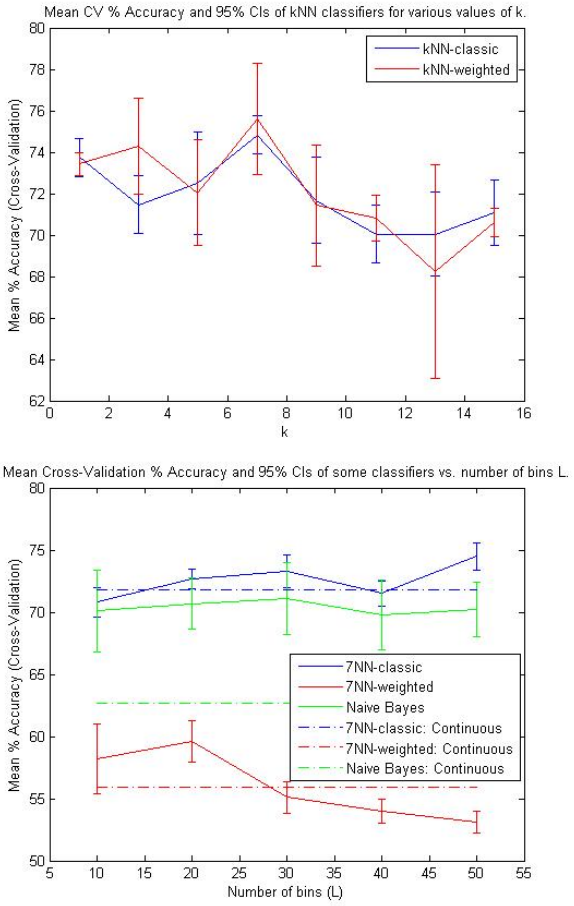

Figure 4. **[Top]** The average CV accuracy with 95% CIs for various choices of k in the kNN Classifiers. **[Bottom]** The average CV accuracy with 95% CIs for various choices of the number of bins L used to discretize the features, for kNN and NB classifiers. Dashed line of the same color represents the average CV accuracy for the same classifier using the original continuous-valued pixel intensities (consult Table 1 to get an idea of what the CIs look like). NB is particularly benefited by the discretization as it now doesn't make the false assumption of Normality of the underlying distributions.

Algorithm 1 AdaBoost using SAMME

1.Initialize the weight distribution D_0 of the training instances to be uniform, i.e. each instance $\boldsymbol{x_i}$ is assigned a weight $w_i = 1/N_{train} \forall i \in \{1, 2, ..., N_{train}\}$.
2.For $t = 1$ to T:
(a) Train a classifier $B^{(t)}(\boldsymbol{x})$ on the training data using weights w_i.
(b) Compute the error of the classifier:
$err^{(t)} = \frac{\sum_{i=1}^{N_{train}} w_i \mathbb{I}(y_i \neq B^{(t)}(\boldsymbol{x_i}))}{\sum_{i=1}^{N_{train}} w_i}$, where $\mathbb{I}(z)$ is the indicator function (assumes value 1 for $z = 1$ and 0 for $z \neq 1$).
(c) Compute the coefficient:
$\alpha^{(t)} = \log(\frac{1-err^{(t)}}{err^{(t)}}) + \log(K + 1)$
(d) Compute the new instance weights:
$w_i \leftarrow w_i e^{\alpha^{(t)} \mathbb{I}(y_i \neq B^{(t)}(\boldsymbol{x_i}))} \forall i \in \{1, 2, ..., N_{train}\}$
(e) Normalize the weights to obtain distribution D_t:
$w_i \leftarrow \frac{w_i}{\sum_{i=1}^{N_{train}} w_i} \forall i \in \{1, 2, ..., N_{train}\}$
3. Classify test instance $\boldsymbol{x_i} \forall i \in \{1, 2, ..., N_{test}\}$ to $C^* = \arg\max_C \sum_{t=1}^{T} \alpha^{(t)} \mathbb{I}(B^{(t)=k}), k \in \{1, 2, ..., K\}$

6. AdaBoost

AdaBoost is a technique that can be used to improve the performance of machine learning algorithms. We generally use *weak learners* (classifiers with low accuracy) as *base classifiers*. As long as their behaviour is not random, *AdaBoost* is guaranteed to offer improved classification results [7]. *AdaBoost* is executed for a series of rounds $t = 1, \ldots, T$. On each round, it trains and tests a new weak classifier. After each round, the distribution of weights $D_t \in \mathbb{R}^N$ assigned to the instances $\boldsymbol{x_i} \in \mathbb{R}^N$ is updated (initially all instances are assigned equal weights). We increase the weight of each incorrectly classified instance and decrease the weight of each correctly classified instance, so that the next classifier focuses on the examples which have so far been misclassified. Here we use our own implementation of the *SAMME* algorithm, proposed in [8], which goes as follows for a general K-class classification problem:

We only used DTs as the *base learners* in our implementation. We run for $T = 20$ rounds (an arbitrary choice, we did not experiment with other values). As we can see in Table 1 the improvement of performance over the *base learners* (*decision trees*) is considerable. Of course the running time of *AdaBoosted DT* is also considerably increased compared to the basic DT classifier. This is particularly apparent when there are many features in the dataset (e.g. using all pixels in the image as features). To decrease running times we could have chosen to use *shallow trees*, i.e. impose some constraint on their *maximum depth*. This would most likely hurt classification performance, but in a real life application we would have to balance between performance and speed.

7. Evaluation

In our case, *accuracy* is an adequate measure for assessing the performance of the classifiers, since all classes are represented by the same number of instances in our dataset and subsequently we expect a uniform class distribution in our test set as well (i.e. the classes are *balanced*). We evaluated the accuracy of every classifier using each different feature set. Each reported accuracy is the mean of 20 independent trials and it is reported with *95% Confidence Intervals*. The results are shown in Table 1. An omission on our part was not including comparisons of the execution times.

Results: *Discretizing* the features improves classification, especially in the case of *NB* where we avoid the dubious Normality assumption. The *feature selection* techniques also improve the results to some extend. Remember, we arbitrarily decided to discard half of the features. These features were not *irrelevant*, simply less *relevant* than the selected ones. Therefore we lost some information. *NB* benefits in the reduced feature sets from the fact that its independence assumption is less unrealistic. This is more pronounced when using the *JMI* criterion rather than *MI*. The DTs already perform the split based on an information-theoretic measure (albeit a different one than *MI*), so the improvement in their case is smaller. *Fisherfaces* and *Eigenfaces* only improve the results in the case of *NB*. *PCA* is a *compression technique* therefore it leads to loss of information. This explains the worst results we get using the PCs as features for the other classifiers. However, the performance of *NB* classifier improved considerably. This happened because on this new feature space the assumption that the features are conditionally independent to one another given the class label is more valid due to the PCs being *orthogonal* to one another. *Fisherfaces* yield better results than *Eigenfaces* for most classifiers, since they are chosen as the directions that best discriminate the data[9], e.g. illumination conditions are ignored when deriving the *Fisherfaces* contrary to what happens in *PCA*. Again, the components are *orthogonal* so *NB* performs well. Finally, *AdaBoosted DT*s improve significantly upon the DTs and offer the best results among all classifiers we examined. *kNN* performs well using pixel intensities as features, with *classic kNN* yielding better results and appearing to be more *robust* (i.e. exhibiting less variance) than *weighted kNN*. DTs' performance was the poorest among all classifiers studied.

[9]Under the assumptions we mentioned.

Table 1. Average Test Accuracy with 95% CIs

Features/Classifier	Classic kNN	Weighted kNN	NB	DT	DT with AdaBoost
Pixel Intensities	0.714 ± 0.013	0.527 ± 0.012	0.623 ± 0.029	0.677 ± 0.013	0.956 ± 0.019
Discretized (20 bins) Pixel Intensities	0.727 ± 0.008	0.596 ± 0.017	0.707 ± 0.021	0.685 ± 0.020	0.914 ± 0.021
Selected Pixel Intensities Discretized (20 bins) by MI (top 50% ranking)	0.816 ± 0.009	0.793 ± 0.009	0.705 ± 0.008	0.691 ± 0.018	0.922 ± 0.019
Selected Pixel Intensities Discretized (20 bins) by JMI (top 50% ranking)	0.846 ± 0.011	0.802 ± 0.012	0.725 ± 0.013	0.697 ± 0.023	0.929 ± 0.019
Eigenfaces (PCA)	0.587 ± 0.012	0.563 ± 0.018	0.822 ± 0.010	0.482 ± 0.019	0.750 ± 0.014
Fisherfaces (LDA)	0.672 ± 0.017	0.625 ± 0.028	0.750 ± 0.019	0.511 ± 0.023	0.782 ± 0.016

8. Conclusion and Suggestions for Future Work

Our results reveal some $(classifier, feature\ vector)$ combinations to work better than others. In other cases the procedure for selecting/ extracting the features introduced assumptions of its own or produced features not in line with the assumptions the classifiers made. In this work, we demonstrated examples of said assumptions not valid in practice. Although reducing the dimensionality of the dataset generally leads to loss of information, still, when done while taking into account domain knowledge about the problem and knowledge of the inner workings of the classification technique used it can improve the classification results. Finally, we observed how *AdaBoost* helps improve *weak learners* such as DTs.

This was not an exhaustive study of face recognition techniques. More classifiers could be included in the comparisons like *Logistic Regression*, *Support Vector Machines*, *Neural Networks*, *classifier ensembles* such as *Random Forrest* or *voting* schemes, possibly using different types of features per classifier and weighted by the individual classifiers *confidence scores*[10].

As for new sets of features, we could consider options widely used in image related tasks such as the ones derived by *Scale Invariant Feature Transform (SIFT)* [9] or *Haar-like filters* [10]. Finally, another approach to dimensionality reduction which unlike *PCA* and *LDA* is not restricted to linear lower-dimensional manifolds is the *Locality Preserving Projection(LPP)* resulting in the so called *"Laplacianfaces"* as features [11].

[10] For the classifiers we examined, a *confidence score* for each prediction is fairly straightforward to calculate. In the case of *classic kNN* it can be the number of nearest neighbours bearing the predicted label divided by k. In the case of *weighted kNN* we could divide the sum of the inverse distances from all nearest neighbours belonging to the predicted class by the total sum of the inverse distances. In the case of DTs we could divide the number of training instances on the leaf belonging to the predicted class by the total number of training instances on that leaf. Finally for *NB* we could directly use the *posterior probability* of the predicted class (i.e. the maximal *posterior*)

9. References

[1]Georghiades, A.S.; Belhumeur, P.N.; Kriegman, D.J. (2001). *From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose*, PAMI 23, pp. 643–660.

[2]Lee, K.C.; Ho J.; Kriegman, D.J. (2005). *Acquiring Linear Subspaces for Face Recognition under Variable Lighting*, PAMI vol. 27, pp. 684–698.

[3]Turk, M.; Pentland, A. (1991). *Face recognition using eigenfaces*, Proc. IEEE Conference on Computer Vision and Pattern Recognition, 1991.

[4]Dhir, C.S.; Iqbal, N.; Soo-Young Lee (2007). *Efficient feature selection based on information gain criterion for face recognition*, ICIA 2007, pp.523–527.

[5]Belhumeur, P.N. ; Hespanha, J. P.; David J. (1997). *Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection*, IEEE Transactions on Pattern Analysis and Machine Intelligence 1997.

[6]Yang, H.H.; Moody, J (1999). *Feature Selection Based on Joint Mutual Information*, In Proceedings of International ICSC Symposium on Advances in Intelligent Data Analysis 1999, pp. 22–25.

[7]Freund, Y.; Schapire, R.E. (1995). A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting.

[8]Zhu J.; Zou, H.; Rosset, S.; Hastie, T. (2009) *Multi-class AdaBoost*, Statistics and Its Interface 2: pp. 349–360, 2009.

[9]Lowe, D.G.; *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, 60, 2 (2004), pp. 91–110.

[10]Viola, P.; Jones, M. (2001)*Rapid Object Detection using a Boosted Cascade of Simple Features*, Conference on Computer Vision and Pattern Recognition 2001.

[11]He, X.; Yan, S.; Niyogi, P.; Zhang, HongJiang (2005). Face Recognition Using Laplacianfaces. IEEE Trans. Pattern Anal. Mach. Intell. 27(3): 328-340.