Project-II by Group Phnom Penh

Christophe Praz
EPFL
christophe.praz@epfl.ch

Daniel Wolfensberger
EPFL
daniel.wolfensberger@epfl.ch

Abstract

In this report, we summarize the results of the second PCML project. The project has two parts and consists of classifying images into two classes first, and then extend the classification to four classes. We first investigated some features transformations and found that computing the square root of the cnn features was improving the test-error so we included this transformation in the model. We then performed a principal component analysis to decorrelate the data and reduce the dimensionality of the problem. We tested five different classification methods: bi/multinomial logistic regression, support vectors machines, neural networks, random forest and naive Bayes classification. We compared the prediction performance of each method with a baseline. We found that SVM with a Gaussian kernel is giving the best accuracy for the binary classification and penalized multinomial logistic regression for the multiclass problem.

1 Data

1.1 Data description

For this project, we were given a set of $N = 6000$ training images, each of them of size 231x231 pixels, color or grayscale. Each image has an associated label $y \in \{1, 2, 3, 4\}$ depending on whether it contains an airplane ($y = 1$), a car ($y = 2$), a horse ($y = 3$) or none of these objects ($y = 4$). We were also provided with a set of describing features $\mathbf{X}_n$ for each image n, the dimensionality of $\mathbf{X}_n$ being $D = 42273$. Among these 42273 input variables, 5408 are Histograms of Oriented Gradients and 36865 have been extracted from a Convolutional Neural Network with the software OverFEAT.

We are tasked to build a classifier which recognizes the object present in a test dataset consisting of $N = 11443$ samples. A matrix of features $\mathbf{X}_{\text{test}}$ with the same dimensionality as the training set is provided, although the images themselves and the output labels $\mathbf{y}_{\text{test}}$ are unknown.

The goal of the project is to produce predictions $\tilde{\mathbf{y}}$ for the test dataset, as well as an approximation of the associated test error. In a first step, we have to do **binary predictions**: predict whether test images contain an object $\in \{$airplane,car,horse$\}$ or not. In a second time, we have to do **multi-class predictions**: predict the type of object present in the test images, ie. predict $\tilde{\mathbf{y}} \in \{1, 2, 3, 4\}$.

It is interesting to see that they are several classification mistakes in the labels $\mathbf{y}$. These mistakes will probably artificially increase our test error. However, it is very likely that these classification mistakes will happen in the test data as well. Consequently, we decided to not correct them to avoid creating a bias between the training and the test sample. As an example, figure 1 below illustrates 3 samples labelled as 4 (other) while clearly containing an airplane, a car and a horse, respectively.

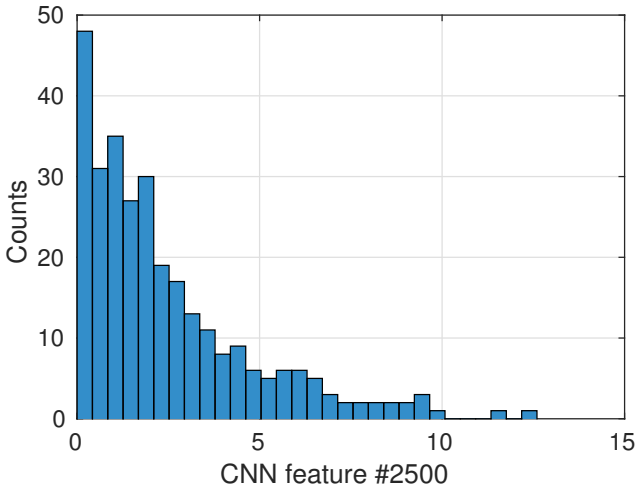
Figure 1: Examples of wrong classification in the labels. The three images have been classified as ”other”.

1.2 Features engineering

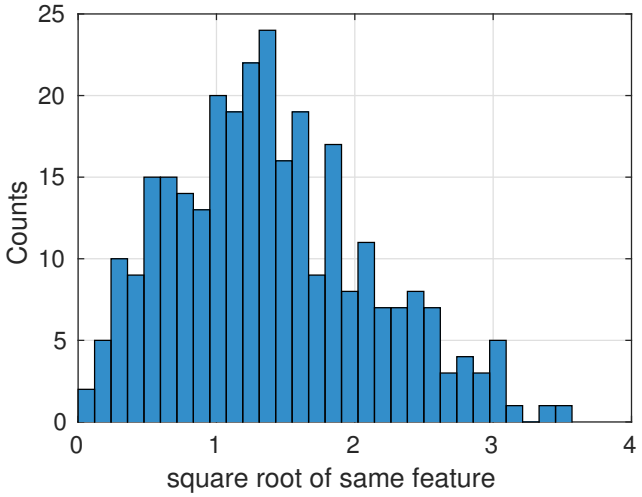
We first split the features matrix $\mathbf{X}$ into the Histograms of Oriented Gradients $\mathbf{X}_{\text{hog}}$ and the features computed with OverFEAT $\mathbf{X}_{\text{cnn}}$ and performed basic exploratory data analysis on each matrix separately.

Histograms of Oriented Gradients are image descriptors used in many computer vision tasks for the purpose of object detection. The technique basically splits the images into several small square cells over which it computes a histogram of gradient orientation quantized by the angle and weighted by its magnitude. For each resulting histogram within a cell (one per orientation), four different normalizations are computed. hog given for this project have been computed using cells of size 17x17 and 8 different orientations, hence the 5408 features ($\lfloor 231/17 \rfloor^2 \times 8 \times 4$). The hog have been computed with a clipping value of 0.2 so that all the values fall within the interval $[0; 0.2]$.

The 36865 columns of $\mathbf{X}_{\text{cnn}}$ features are all positive, varying between 0 and 65 for the largest feature. The matrix is very sparse with more than 95% of the entries equal to zero. Looking at the histograms of the cnn features, one can see that many of them show the same pattern with a large peak of zero entries followed by a long tail extending between 0 and the range of the feature. Figure 2(a) illustrates one of these distribution (feature $\#2500$). This tailed distribution shape suggested us to test different feature transformations to make it more symmetric or "gaussian-like", such as $\sqrt[n]{\mathbf{X}_{\text{cnn}}}$ or $\log\left(1 + \mathbf{X}_{\text{cnn}}\right)$. As an example, figure 2(b) shows the distribution of the square root of the feature displayed on the left panel. One can see that the distribution is more compact and looks more normal, which is a desired property for many classification methods.



(a) Typical distribution of one of the cnn features. Zero entries are not shown.



(b) Distribution of the square root of the same feature.

Figure 2:

On that basis, we analyzed the impact of diverse feature transformations on the test-error and found that the square root $\sqrt[2]{\mathbf{X}_{\text{cnn}}}$ was significantly improving the performance of most classification methods, with an average test BER being $\sim 1\%$ better than when using the raw features matrix $\mathbf{X}_{\text{cnn}}$.

1.3 Dimensionality reduction

The very high dimensionality of the features space and more particularly the fact that $D \gg N$ encouraged us to consider dimensionality reduction methods. We decided to apply Principal Component Analysis to the data for multiple reasons. First, because $D > N$, PCA will reduce the dimensionality of the features space to a maximum of $D_{pca} = N$ and thus significantly improve the computational time of the classification methods. Moreover, keeping only these components is sufficient to explain all the variability contained in the initial high dimension features matrix. It is like a lossless compression. It does not guarantee that the extracted components are as good as the raw features to explain the output labels though, this has to be tested. Secondly, PCA will decorrelate the initial feature vectors and therefore notably help to reduce ill-conditioning. In this respect, PCA can be seen as a regularization technique. Finally, since the principal components come ordered (the ones explaining the most of the variability coming first), one could try to keep only a reduced amount of components and still expect to explain a considerable part of the variability contained in the features.

We tried to apply PCA on the whole matrix $\mathbf{X}$ but also on $\mathbf{X}_{\text{hog}}$ and $\mathbf{X}_{\text{cnn}}$ separately. It appeared that for all the classification methods we tested, they were performing as good or better when considering only the cnn features, suggesting that the hog features were not adding any new information relevant for the classification task. To test this assumption, we performed a linear regression of the 3 principal components of the hog using all the principal components of the cnn. It revealed that these three hog components could be written as a linear combination of the cnn within a tolerance interval of 10^{-4}. This does not represent a solid proof but explains partially why we did not see any improvement adding the hog features or not.

As a result, we performed a PCA on $\sqrt[2]{\mathbf{X}_{\text{cnn}}}$ only. Figure 3 shows 2d scatterplots of the 3 resulting principal components. One can see that with these 3 features only we can separate the data quite well between the 4 classes, although they all regroup at the center of the plots were the class "other" is located. Datapoints located in this area will probably be harder to classify. Some of the black dots ("other") are located quite far on the branches of the other classes. These points might be explained by the classification mistakes in the training labels pointed out in section 1.1.

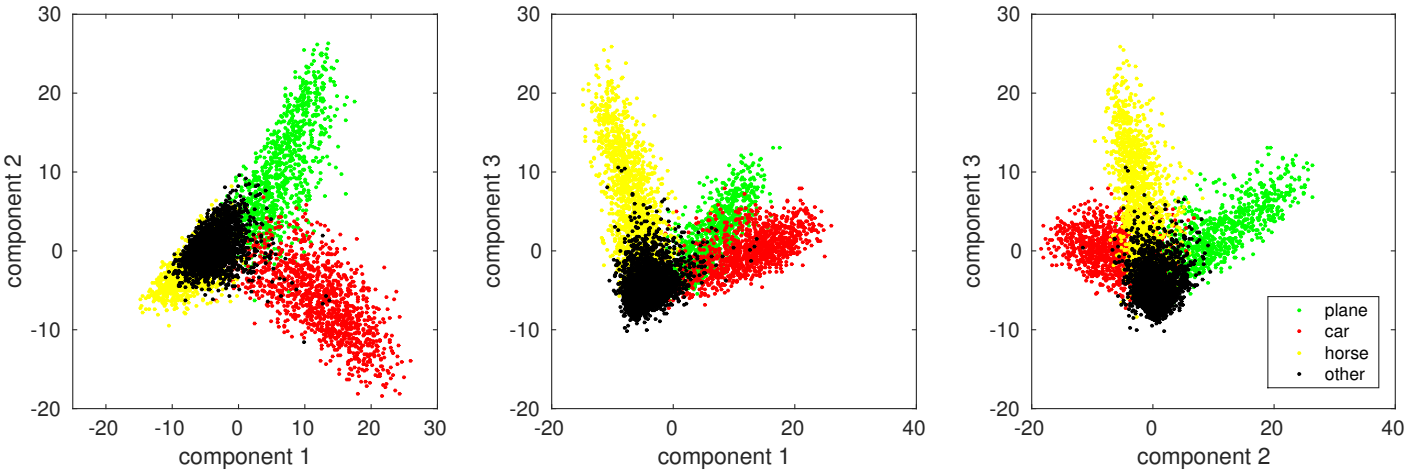


Figure 3: Scatterplots of the 3 principal components extracted from square root of the CNN features.

In general, it is good practice to normalize the features before performing a PCA, mainly to avoid that features varying on a larger scale will dominates the projected principal components (since PCA is maximizing the variance of the projections). However, we noticed that normalizing the cnn features in our case was resulting in worse classification performance. Although we do not know in details how the cnn features are extracted, we do believe that their respective range is adding some relevant information, which would explain why normalizing them is not a good idea.

In conclusion, we first discarded the hog features and applied a square root transformation on the cnn features. We then centered (but not normalized) the features and performed a PCA. The resulting matrix $\mathbf{X}_{\text{pca}}$ of dimension 6000×6000 is the input we used for all the classification methods explained in Section 2.

2 Methods description

Five classification methods were tuned, tested and compared with a baseline, for both binary and multiclass classification. Only the four methods that performed better than the baseline will be detailed below, the fifth being Naive Bayes classification, which performed very poorly (average test BER around 50%), which can be explained by the fact that this methods assumes features independence which is very unlikely for spatial data such as images.

2.1 Reference baseline

In order to assess the increase in performance brought by more advanced classification technique, we built a simple baseline model. This baseline corresponds to the most trivial yet relevant way of classifying new images given a set of D features from a training set. It simply assigns a given point to the class c whose centroid is the closest. Hence for a given point i:

$$c_i = \underset{c}{\operatorname{argmin}} \sum_{j=1}^{D} \left(x_{i,j} - \frac{1}{N_c} \sum_{m \in c} x_{m,j} \right)^2$$

where $\frac{1}{N_c} \sum_{m \in c} x_{m,j}$ is the mean of the feature j for all training points in class c.

2.2 Bi/multinomial logistic regression

Binomial logistic regression can be seen as a particular case of multinomial logistic regression so only the later will be described. In multinomial logistic regression, the probability $p(y_i = c|\tilde{\boldsymbol{X}}_i)$ of a given point i to belong to a given class $c = 1, ..., K$ takes the following form:

$$h_{\boldsymbol{\beta}}(\tilde{\boldsymbol{X}}_{\boldsymbol{i}}) = \begin{bmatrix} p(y_i = 1|\tilde{\boldsymbol{X}}_i, \boldsymbol{\beta}) \\ p(y_i = 2|\tilde{\boldsymbol{X}}_i, \boldsymbol{\beta}) \\ \vdots \\ p(y_i = K|\tilde{\boldsymbol{X}}_i, \boldsymbol{\beta}) \end{bmatrix} = \frac{1}{\sum_{k=1}^{K} \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(k)}\right)} \begin{bmatrix} \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(1)}\right) \\ \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(2)}\right) \\ \vdots \\ \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(K)}\right) \end{bmatrix}$$

where $\boldsymbol{\beta}^{(1)}$, $\boldsymbol{\beta}^{(2)}$,...,$\boldsymbol{\beta}^{(K)}$ are the parameters of the model, hence in multinomial logistic regression $\boldsymbol{\beta}$ is a D-by-K matrix, where D is the number of features and K the number of classes. Note that the division by $\sum_{k=1}^{K} \exp(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(k)})$ normalizes the distribution so that it sums to one over all classes.

It can be shown that the cost function for multinomial logistic regression (i.e. the negative of the log-likelihood) is

$$J(\boldsymbol{\beta}) = -\left[\sum_{i=1}^{N}\sum_{k=1}^{K} \{y_i = k\} \log \frac{\exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(k)}\right)}{\sum_{j=1}^{K} \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(j)}\right)}\right]$$

It is not possible to solve for the minimum of $J(\boldsymbol{\beta})$ analytically, so an iterative optimization algorithm such as gradient descent is needed. By taking the derivatives of $J(\boldsymbol{\beta})$ as a function of $\boldsymbol{\beta}$, the resulting gradient is

$$\nabla_{\boldsymbol{\beta}^{(k)}} J(\boldsymbol{\beta}) = -\sum_{i=1}^{N} \left[\tilde{\boldsymbol{X}_i^T}\left(\{y_i = k\} - p(y_i = k|\tilde{\boldsymbol{X}}_i, \beta)\right)\right]$$

which is a vector of dimension D. This means that the overall gradient is indeed a D-by-K matrix just as $\boldsymbol{\beta}$. Once the cost has been optimized and the corresponding matrix $\boldsymbol{\beta}_{\text{min}}$ has been determined, any new point i can be classified by:

$$c_i = \underset{c}{\operatorname{argmax}} \frac{\exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(c)}\right)}{\sum_{k=1}^{K} \exp\left(\tilde{\boldsymbol{X}_i^T}\boldsymbol{\beta}^{(k)}\right)}$$

It is recommended to add a regularization term in the cost function of the logistic regression in order to avoid overfitting and make convergence possible even if the groups are linearly separable in the features space. In our case, we chose the classical regularization term λ:

$$J_{\text{reg}}(\beta) = \lambda \sum_{j=1}^{D}\sum_{k=1}^{K} \left(\beta_j^{(k)}\right)^2$$

As such λ is the only intrinsic parameter of the logistic regression, the other parameters being used for the tuning of the gradient descent method (c.f. Section 2.7).

2.3 Support Vectors Machine

Support Vectors Machine (SVM) is a non-probabilistic binary classifier which optimizes the following cost function :

$$\underset{\beta}{\operatorname{argmin}} \sum_{n=1}^{N} C[1 - y_n \tilde{\mathbf{X}}^T \boldsymbol{\beta}]_+ + \frac{1}{2}\sum_{j=1}^{M} \beta_j^2$$

Without going into details here, one can show that this cost can be optimized by solving the Lagrangian dual of the above problem. One main advantage of SVM is that the the resulting dual vector $\boldsymbol{\alpha}$ is sparse, and only non-zero entries of the training data are required to determine the decision boundary, thus the predictions $\tilde{\mathbf{y}}$. This remarkable property is particularly profitable for problems dealing with large matrix and computational constraints like it is the case here. Another noteworthy property of SVM is that the dual problem appears naturally kernelized. In other words, we can apply the method on nonlinearly transformed features $\boldsymbol{\phi}_n$ without having to specify directly the nonlinear transformation $\boldsymbol{X}_n \rightarrow \boldsymbol{\phi}_n$. We just need to define the kernel $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$.

There are two important parameters in this method : The *margin* C which can be seen as the inverse of a regularizer like λ in Ridge regression. The larger the margin, the more the samples located on the "wrong" side of the decision boundary are penalized, giving the model freedom to select more samples as support vectors (more complex model). As a result, large margins increase the risk to overfit the training data. Smaller margins correspond to smoother decision boundary and more regularized models. The second parameter is the choice of the *kernel function* and the parameter(s) included.

2.4 Neural networks

Neural networks can be roughly seen as a series of layers of nested logistic regressions, where the outputs of one layer are the inputs of the next. However unlike logistic regression the activation function is not necessarily a sigmoid. The main parameters of such a network are the *number of layers of neurons* (nodes) between the input and the output layer (hidden layers) as well as the *number of neurons in each of the hidden layers*. Two other important parameters are related to the optimization of the weights at every neuron through the back-propagation algorithm: the *number of epochs* determines the number of times each training vector is sequentially passed through the network and the weights are updated while the *mini-batch size* corresponds to the number of samples that are used to compute the gradient in the back-propagation algorithm.

2.5 Random forest

Random forests are ensembles of decision trees. To generate the individual trees the algorithm selects a random sample of data with replacement and fits the tree by using these samples only, a technique known as tree bagging. Another specificity of this method is that during the generation of the individual decision trees, a random subset of the features is used at each candidate split. This is done in order to uncorrelate the individual trees and to reduce the overall estimation variance.

There are three critical parameters in this method: the *number of individual trees* that are being generated, the *minimum number of samples* that are required to perform a new split (size of the leaf), and the *number of features to select at random* for each decision split.

2.6 Performance evaluation

To evaluate the performance of the different classification method we used the Balanced Error Rate (BER) score:

$$\text{BER} = \frac{1}{C}\sum_{c=1}^{C}\left[\frac{1}{N_c}\sum_{n=1}^{N}\{y_n = c\}\{y_n \neq \hat{y}_n\}\right]$$

where C is the number of classes, N_c the number of samples in class c, y_n, the ground truth for sample n and $\hat{y}_n$ the predicted class. The balanced error rate can be seen as the average of the errors in each class, which means that all classes will contribute identically to the total error even if the classes are not equally represented. This is different from the overall accuracy which in case of class imbalance will be dominated by the error in the dominant class.

2.7 Implementation details

Dimensionality reduction: We first split between train and test samples, then we center the train data, compute the PCA over the train data, keep in memory the change of basis matrix, center the test data around the centroids of the train data and eventually compute the principal components of the test data with the change of basis matrix. It is important not to take into account the test data at all while performing the PCA in order to avoid biasing the model.

Binarization: To perform binary classification, we simply binarized the training labels y (1,2,3 $\rightarrow$ 1 and 4 $\rightarrow$ 0). We tried other methods (binarizing after multiclass classification or 1-vs-all classification to get the most likely class) but the performance was similar or worse.

Random forest: We used the standard implementations from MATLAB, *TreeBagger*. Note that for this method we decided to only use the first 100 components of the PCA, because we were getting similar performance for a much faster computational time. As a result, we decide to take into account all the features (ie. 100) for each decision split. As the components do not have the same weight after PCA, it is important to avoid discarding the few most important before performing a split.

Neural network: We used the implementation from the DeepLearnToolbox for MATLAB by Rasmus Berg Palm.

Bi/multinomial logistic regression: For binomial logistic regression, we started with our implementation from Projet-1 and improved its scalability by using mini-batch gradient descent. By default a mini-batch size of 100 samples is used, but this can be changed in the function call. To make the convergence faster, we used a momentum parameter in the update of the β parameters. Thus at iteration the i:

$$\beta_i = \alpha\,(g_i + \mu\, g_{i-1})$$

where μ is the momentum parameter and g_i is the gradient at iteration i . Since the objective function is quite smooth, we could use a high value of $\mu = 0.9$.

In terms of performance the critical part of the algorithm is the evaluation of the cost function (negative of log-likelihood). Thus, the cost is evaluated only every 100 iterations.

We increased the regularization that is already obtained by the PCA by adding an additional penalization term of $\lambda = 0.01$, which is sufficiently small not to affect the performance but allow convergence in case of perfect separation.

Since we are also dealing with a multiclass classification, we implemented multinomial logistic regression, using a similar structure to the binomial logistic regression algorithm with the same mini-batch and momentum gradient descent. The parameters of both methods are the same as well.

Support vectors machine: We implemented our version of SVM based on the sequential minimal optimization algorithm given for the lab on SVM. We tried 3 different kernels in this study : linear, Gaussian and Lorentz. These kernels are explicited below :

$$\begin{aligned} \mathbf{K}_{\text{Linear}} &= \mathbf{\Phi}\mathbf{\Phi}^T \\ \mathbf{K}_{\text{Gaussian}} &= \exp\left(-\gamma||\boldsymbol{\phi}_i - \boldsymbol{\phi}_j||^2\right) \\ \mathbf{K}_{\text{Lorentz}} &= \exp\left(-\gamma||\boldsymbol{\phi}_i - \boldsymbol{\phi}_j||\right) \end{aligned}$$

where γ is a parameter of the exponential kernels and can be related to the "area of influence" of the training samples. Smaller values of γ corresponding to a broader area of influence.

Although SVM is a binary classifier, it can be extended to multiclass classification. There are traditionally two ways of doing it. In the the method called "one versus all", we iteratively build a classifier between a class c_i and the rest of the samples. Each test sample is then attributed to the class which obtained the best score. In the method called "one versus one", we train $K(K-1)/2$ binary classifiers for a K multiclass problem. For prediction, each classifier is applied successively to the test sample and the class which was the most often assigned to the sample is kept.

Models parameters tuning: In order to tune the parameters, we first started exploring the parameter space following a trial and error procedure to get some insight on the sensibility and impact on the model of each parameter (random search). We then performed grid search around values that gave good results in order to fine tune the parameters. We adjusted the parameters using cross-validation, choosing the parameters giving the best average testing BER and the less variance around it. Note that the dimensionality reduction process was essential here to reduce4heur the computational time and be able to perform precise grid search. Table 1 summarizes the parameters used for each of the classification methods.

Model	Tuned parameters
Logistic regression	λ = 0.01, μ = 0.9, $\alpha = 10^{-5}$, mini batch size = 100 samples
Random forest	$N_{\text{trees}} = 100$, leaf size min. = 3, numFeaturesToSample = all
Neural network	nb. hidden layer = 1, nb. hidden neurones = 100, mini batch size = 100 samples
SVM	margin $C = 100$, Gaussian Kernel, $\gamma = 10^{-6}$

Table 1: Values of the models' parameters used to produce results shown in section 3.

3 Results

3.1 Binary classification

We assessed the performance of the different methods with cross-validation. We chose 4-fold cross-validation as it seemed to be a good trade-off between a large amount of data for training and enough samples to have a reliable test error. In order to evaluate the average BER and its variability, we performed 10 iterations of 4-fold cross-validation, resulting in a total of 40 train and test errors per method. Boxplots of the test error for each method are reported on figure 6(a). One can see that applying the baseline on the principal components reduces the average BER by $\sim 15\%$, compared with the full data. The 4 methods selected performed even better with an average BER $< 10\%$ and a limited variance. SVM with $\mathbf{K}_{\text{Gaussian}}$ seems to give the best results with an average BER of 7.68 $\pm$ 1.32 $\%$. The $\pm$ interval corresponds to the standard deviation computed over all cross-validation instances (i.e. 4×100 BER). However, the difference with logistic regression and SVM with a linear kernel (not shown here, average BER of 7.98$\%$) might not be significant. Compared with the baseline, most of the improvement in the BER brought by the 4 methods seems to be related to a strong decrease in the fraction of false negatives (e.g. 4.1% for SVM vs 19.27% for the baseline).

In order to further assess the performance of the classifiers, we built a Receiver Operating Characteristic (ROC) curve for the three best methods (SVM, NN and Logistic regression) and the baseline (Figure 4). The curves are created by plotting the fraction of true positives against the ratio of false positives at various decision threshold settings. As a result, a completely random classifier would give a diagonal straight line whereas a perfect classifier would jump directly from a true positive rate of 0 to 1 as we reduce the threshold and follow the upper left corner of the plot. One can see that the 4 models displayed perform well according to the ROC curves. Here as well, SVM seems to give the best performance with the highest area under the curve (AUC). It has also the steepest slope, meaning that it is better at maximizing the true positive rate while minimizing the false positive rate.

Finally, we computed a learning curve for the best method (i.e. SVM Gaussian kernel with hyperparameters as described in Table 1). We first held out 25$\%$ of the data that we kept as a fixed test sample. Then, we iteratively increased the size of the training set from 10$\%$ to 90$\%$ of the rest of the dataset. For each proportion of the training data, we did 10 iterations of random training data sampling to get insight on the variance of the test error. We see on Figure 5(a) (a) that the test BER decreases quickly first and seems to reach a plateau as we increase the training set size, suggesting that we are using enough data for the model to converge. The shaded areas also indicates a slight reduction of the error variance as we increase the train sample, even though it is not obvious. Note that the

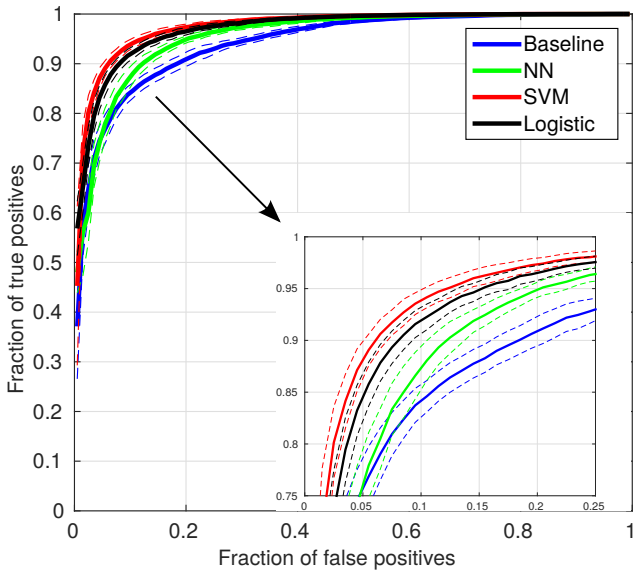
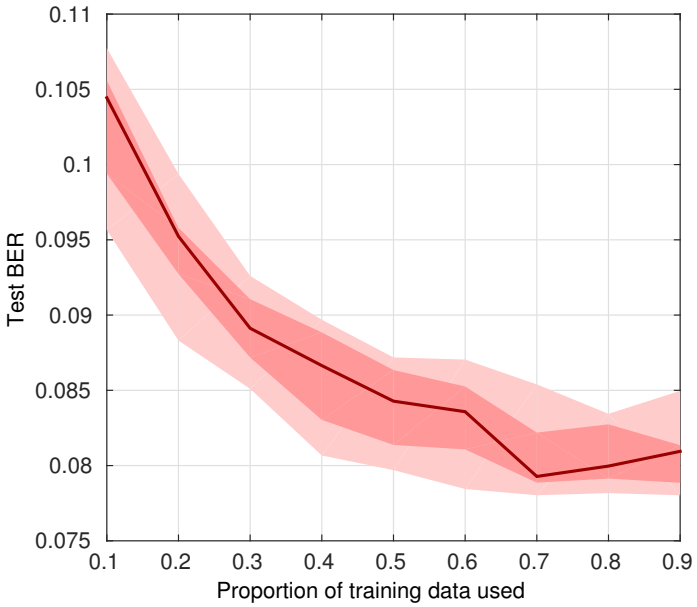
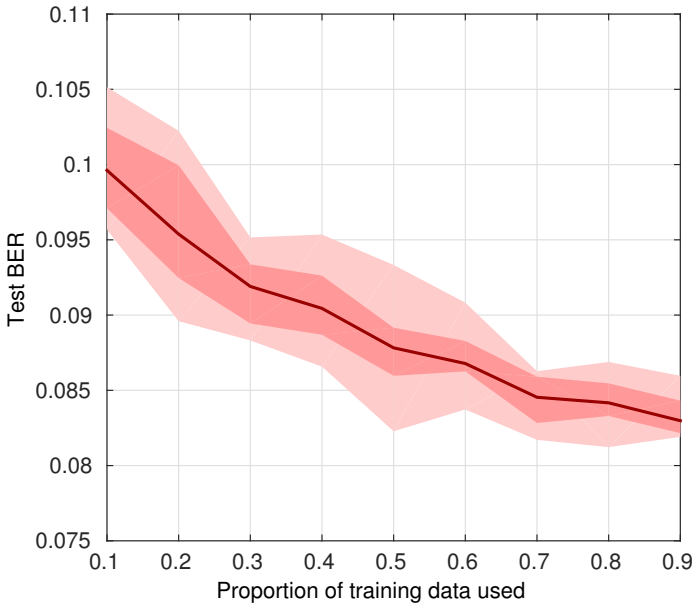


Figure 4: ROC curves for the binary classification problem. The bold line shows the mean and the distance between the dashed lines the standard deviation. The small subpanel is a zoom on the top left corner of the image. First we sampled many points (pair of TP/FP values) by varying the decision criterion of every method between 0 and 1 (the standard decision criterion being 0.5). Curves were then generated by creating 100 horizontal bins (along the x-axis) and averaging all the points inside them.



(a) SVM with Gaussian kernel - Binary problem.



(b) Penalized multinomial logistic regression - Multiclass problem.

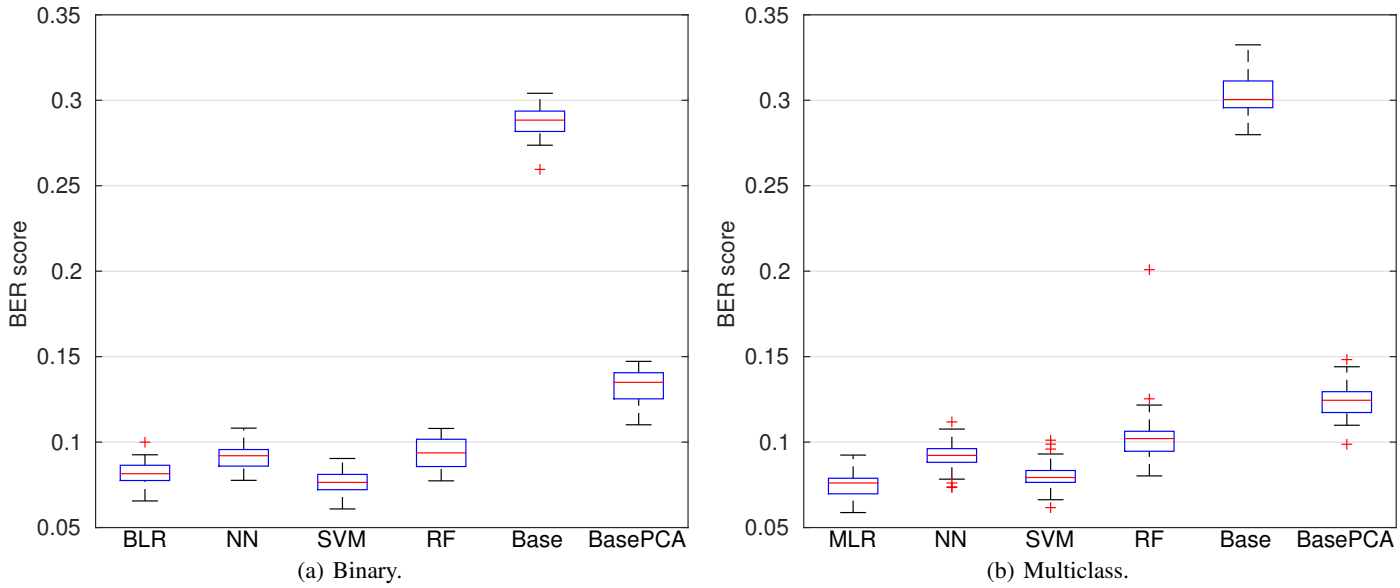
Figure 5: Learning curves computed for each problem best method. Dark areas correspond to the 25-75 percentile range whereas light areas correspond to the 10-90 percentile range. The bold line is the median.

average BER displayed is slightly above its value on the boxplot (fig. 6(a)). This is because it has been computed over a unique test sample here.

We decided not to show any statistics about the training error, the main reason for it being that these errors are in most cases zero or very close to it. Even though a null classification error on the train sample might be a sign of overfitting, we observed that by increasing the regularization of our models (λ for logistic regression, C for SVM), we were getting a higher test error as well.

3.2 Multiclass classification

Similar to what we have done for binary classification, we performed 10 iterations of 4-fold cross-validation for each of the multiclass methods. Boxplots of the resulting test BER are shown on figure 6(b). We did not show the results for the SVM 1-vs-1 method as they are very similar but slightly worse than for 1-vs-all. In general, the same conclusions as for the binary case can be drawn except that in the present case multinomial logistic regression seems to give a slightly better performance than SVM, with a BER of 7.65 ± 1.56 %. The learning curve of multinomial logistic regression is shown in Figure 5(b) (b). It behaves quite similarly to the SVM curve, except that the decrease in variance is stronger and the trend is more linear.



(a) Binary. (b) Multiclass.

Figure 6: Comparison of the different methods applied to the binary and multiclass problem. Blue boxes represent the 25-75 quartiles, red lines are the medians, the whiskers extend to 1.5 $\times$ the interquartile range and red crosses are outliers. BLR stands for binomial logistic regression, NN for neural network, SVM for SVM with a gaussian kernel, RF for random forest, Base for the baseline and BasePCA for the baseline performed on the principal components.

In complement to the boxplots, one can see on Table 2 an averaged confusion matrix for the multiclass problem. The left table shows the baseline accuracy while the right panel represents our best method, ie. multinomial regression. It is interesting to see that the baseline has very limited skill at classyfing between "Horse" and "Other", which might be linked to the figure 3 where we see that the class "Horse" is the less separated from the the "Other" data. The multinomial logistic regression is doing significantly better in general, with the exception of the "Other" wrongly classified as "Plane" where the improvement is limited. This could be partly explained by the mistakes in the labels emphasized in section 1.1.

		Real [%]			
		Plane	Car	Horse	Other
Pred. [%]	Plane	12.50	1.74	0.61	1.22
	Car	1.92	14.86	0.63	1.96
	Horse	1.28	0.26	11.90	11.43
	Other	2.77	0.90	5.64	30.38

		Real [%]			
		Plane	Car	Horse	Other
Pred. [%]	Plane	14.86	0.13	0.04	1.03
	Car	0.11	18.62	0.04	0.60
	Horse	0.05	0.02	22.07	2.72
	Other	0.55	0.50	1.95	36.7

Table 2: Confusion matrix (in percentage) for multiclass classification for the baseline (left) and multinomial logistic regression (right)

3.3 Predicting the test data

To predict the test data, we first centered the train data and performed a PCA. We then centralized the test data around the centroids of the train data and computed the components of the test data by multiplying with the matrix of coefficients associated with the PCA of the train data (the change of basis matrix). We then trained the best classification methods (SVM with gaussian kernel for binary, penalized multinomial logistic regression for multiclass) with all the training data to compute the predictions on the test samples. An estimation of the test error is given in the next section.

4 Summary

We developed and tested various methods to do binary/multiclass image classification. For the binary problem, we found that SVM with a tuned Gaussian kernel is showing the best performance in terms of BER. We estimated that the associated test BER is **7.68** $\pm$**1.32** $\%$. Concerning the multiclass problem, we found that the penalized multinomial logistic regression was the best method with a test BER of **7.65** $\pm$**1.56** $\%$. The $\pm$ interval corresponds to the standard deviation computed over all cross-validation instances.

Acknowledgments

All the work presented in this report was done by Christophe and Daniel. We were in constant interaction during the project and wrote this report together. We would like to thank Emti for the report sample which served as a model for the structure of our report.